University of Strathclyde

Department of Electronic and Electrical Engineering

Secure Decentralised Storage Networks

by

Greig Paul

A thesis submitted in fulfilment of the

requirements for the degree of

Doctor of Philosophy

to the University of Strathclyde

Submitted for Examination, April 2017

Declaration

This thesis is the result of the author's original research. It has been composed by the author and has not been previously submitted for examination which has led to the award of a degree.

Signed:

Date: October 5, 2017

“*Security is a process, not a product.*”

Bruce Schneier

Abstract

In recent years, cloud-based computing and storage have become increasingly popular, as they remove the need for users and developers to buy or rent expensive dedicated hardware on an ongoing basis. This has led to the increasing centralisation of both services and storage, where users are reliant upon a small number of cloud-based providers to hold their data, and provide them with services they use. Recent events have shown that security breaches of centralised data stores can lead to significant quantities of personal data being revealed. This centralisation can also result in inconvenience in the event of the failure of the service provider, resulting in potential data loss or a loss of utility of the service.

In contrast, a decentralised service and storage architecture removes the single point of failure from a network, and allows users to remove their dependency on a single company or service provider. In addition, by preventing storage providers from having access to user data, as is inherently needed in a decentralised network to preserve confidentiality, it is possible for users to protect their data from theft or unauthorised access, giving rise to data security and privacy benefits.

This thesis explores the the challenges encountered in implementing a secure decentralised network, based around storage, and presents solutions to some of these problems. A security analysis of the MaidSafe network is firstly given, setting the context of the work, and investigating the state-of-the-art. Potential uses for decentralised services are considered, including for use on mobile devices. The importance of client device security is also considered, and a number of vulnerabilities affecting the security of client-based software are identified and explored. A practical design of decentralised architecture for preserving user privacy when discovering users is also contributed, to illustrate how decentralised service design can be used to enhance privacy of existing systems, and solve otherwise unsolved problems. A review and analysis of the privacy policies of popular web-based services then shows the extent to which user privacy is at risk from centralised web services. Finally, the concepts of identity and authentication within decentralised networks are considered, with a novel smartcard-based approach to securing user credentials within a decentralised network demonstrated.

Acknowledgements

I would like to extend my thanks to many people, without whom I would not have been able to complete this work. My family, friends and colleagues, in particular James Irvine, Xavier Bellekens, and Ross McPherson.

Contents

List of Publications

[A1] G. Paul, F. Hutchison, and J. Irvine, “Security of the MaidSafe Vault Network,” in *Proceedings of Wireless World Research Forum Meeting 32*, May 2014.

[A2] G. Paul and J. Irvine, “Privacy implications of wearable health devices,” in *Proceedings of the 7th International Conference on Security of Information and Networks*, SIN ’14, (New York, NY, USA), pp. 117:117–117:121, ACM, 2014.

[A3] 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*, SIN ’14, (New York, NY, USA), pp. 69:69–69:72, ACM, 2014.

[A4] G. Paul and J. Irvine, “Google’s Android setup process security,” in *Proceedings of Wireless World Research Forum Meeting 33*, September 2014.

[A5] X. Bellekens, G. Paul, J. Irvine, C. Tachtatzis, R. Atkinson, T. Kirkham, and C. Renfrew, “Data remanence and digital forensic investigation for CUDA graphics processing units,” in *Proceedings of 1st IEEE/IFIP Workshop on Security for Emerging Distributed Network Technologies (DISSECT)*, 2015.

[A6] G. Paul and J. Irvine, “Achieving optional Android permissions without operating system modifications,” in *Proceedings of Vehicular Technology Conference (VTC Spring), 2015 IEEE 81st*, 2015.

[A7] G. Paul and J. Irvine, “5G-enabled decentralised services,” in *Proceedings of Vehicular Technology Conference (VTC Spring), 2015 IEEE 81st*, pp. 1–5, IEEE, May 2015.

[A8] G. Paul, P.-L. Dubouilh, and J. Irvine, “Performance challenges of decentralised services,” in *Proceedings of Vehicular Technology Conference (VTC Fall), 2015 IEEE 82nd*, pp. 1–4, September 2015.

[A9] G. Paul, D. Thomas, and J. Irvine, "Privacy implications of smartphone-based connected vehicle communications," in *Proceedings of Vehicular Technology Conference (VTC Fall), 2015 IEEE 82nd*, pp. 1–3, September 2015.

[A10] G. Paul and J. Irvine, "Take control of your pc with UEFI secure boot," *Linux Journal*, no. 257, pp. 58–72, 2015.

[A11] D. Thomas, G. Paul, and J. Irvine, "Going beyond the user — the challenges of universal connectivity in IoT," in *Proceedings of Wireless World Research Forum Meeting 35*, 2015.

[A12] P.-L. Dubouilh, G. Paul, and J. Irvine, "Performance of WebRTC in the context of a decentralised storage solution," in *Proceedings of Wireless World Research Forum Meeting 35*, 2015.

[A13] G. Paul and J. Irvine, "Automating identification of potentially problematic privacy policies," in *Proceedings of Wireless World Research Forum Meeting 35*, 2015.

[A14] G. Paul and J. Irvine, "Practical attacks on security and privacy through a low-cost Android device," *Journal of Cyber Security and Mobility*, 2016.

[A15] G. Paul and J. Irvine, "IEDs on the road to fingerprint authentication: Biometrics have vulnerabilities that PINs and passwords don't," *Consumer Electronics Magazine, IEEE*, vol. 5, pp. 79–83, April 2016.

[A16] N. Mathur, I. Glesk, A. Davidson, G. Paul, J. Banford, J. Irvine, and A. Buis, "Wearable mobile sensor and communication platform for the in-situ monitoring of lower limb health in amputees," in *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*, May 2016.

[A17] G. Paul and J. Irvine, "Automating identification of potentially problematic privacy policies," *Nordic and Baltic Journal of Information and Communications Technologies*, 2016.

[A18] X. Bellekens, G. Paul, C. Tachtatzis, J. Irvine, and R. Atkinson, "Strategies for protecting intellectual property when using graphics processing units for CUDA applications," in *Proceedings of the 9th International Conference on Security of Information and Networks*, SIN '16, (New York, NY, USA), ACM, 2016.

[A19] D. Thomas, R. McPherson, G. Paul, and J. Irvine, "Optimizing power consumption of Wi-Fi for IoT devices: An MSP430 processor and an ESP-03 chip provide a power-efficient solution," *Consumer Electronics Magazine, IEEE*, 2016.

[A20] N. Mathur, G. Paul, J. Irvine, M. Abuhelala, A. Buis, and I. Glesk, "Towards a low cost platform for remote monitoring of lower limb health of amputees in the developing world," *IEEE Access*.

[A21] G. Paul and J. Irvine, "Investigating the security of Android security applications," in *Proceedings of the 9th CMI Conference on Smart Living, Cyber Security and Trust*, November 2016.

[A22] D. Broby and G. Paul, "Blockchain and its use in financial settlements and transactions (to appear)," *Journal of the Chartered Institute for Securities and Investment (Review of Financial Markets)*.

Table of Abbreviations

API Application Programming Interface

APT Advanced Persistent Threat

BGP Border Gateway Protocol

CA Certificate Authority

DHT Distributed Hash Table

DNS Domain Name System

DRM Digital Rights Management

GDPR European General Data Protection Regulation

GMS Google Mobile Services

HTTPS Hypertext Transfer Protocol (Secure)

ICANN Internet Corporation for Assigned Names and Numbers

MDM Mobile Device Management

MITM Man in the Middle attack

RAT Remote Access Trojan

SSL Secure Socket Layer

TLS Transport Layer Security

Chapter 1

Introduction

In recent years, computer security incidents have risen in status from non-events to headline news. Barely a day goes by without high-profile news reports of a significant security breach or incident of data theft from internet-connected systems, including major household names with international recognition. It is now difficult to name a large company which has not suffered from a major, news-worthy security breach. Adobe [1], Microsoft [2], Sony [3, 4], Target [5], Anthem [6], the US Office for Personnel Management (OPM) [7], Dropbox [8], Experian [9, 10] and Ashley Madison [11], amongst others, have suffered from either cyber-attacks or significant data breaches in recent years, many of which exposed or made available significant quantities of personal data.

Much of the data held by companies can be considered sensitive, either by legal definition, such as medical or healthcare related data, or by nature of the context from which it was retrieved. For example, while a security incident involving the release of email addresses may not necessarily appear at first glance to be particularly severe, users may feel this a much more serious breach if it were an email list of those attending a particular clinic for a transmittable or socially stigmatised health condition [12].

Despite data protection legislation such as the UK Data Protection Act [13] being in force and providing protections for the rights of data subjects, security breaches resulting in the exposure of personal information continue to happen, on an alarming scale. Meanwhile, targeted attacks by foreign threats, commonly referred to as Advanced Persistent Threats (APTs), present an ongoing a risk of intellectual property theft or unauthorised access to sensitive data by external attackers [14].

Despite the diverse range of threats being faced by companies and services, one common factor in many data breaches is the lack of encryption of data held on the server. Where encryption is used, such as with services like Dropbox, often the encryption operates at storage or server level. This means that data is stored encrypted on the physical disks, but that user data is accessible in decrypted form to the whole application and service [8]. In situations such as these, the encryption only offers protection against offline attack, where the servers of the service provider are examined while powered off (and thus no keys are held in memory), or where physical disks are imaged and exfiltrated without the corresponding keys. When powered on, and connected to the application, they expose data in decrypted form. Transparent server-side encryption systems such as these remove the benefit of the encryption, and instead security falls to that of the lowest common denominator, usually the user account login process, since unencrypted user data will be made available to a logged-in session. This was seen in the case of Dropbox, where login validation logic was found to be non-functional, and user data was potentially exposed to the wider internet [8].

It is certainly possible that the link between companies not employing encryption across data, and data breaches, is indirect — one could reasonably conclude that companies employing encryption may be spending more money on a security team, and implementing their recommendations, thus resulting in a strong culture of security within the organisation and a more secure product overall, which incorporates security as a design feature. Nonetheless, encryption offers a secondary layer of protection against security breaches — even if an insecure service is breached, if data is encrypted in a manner where the cryptographic keys are not accessible to the service operator, the user data itself remains secure. This defence-in-depth inspired approach is one rarely seen, and where it is employed, it is often used as a unique selling point of a service, such as Spideroak[1], rather than as a routine, best-practice security measure. Indeed, this scenario is one addressed by the European Union's upcoming General Data Protection Regulation [15], which offers an exception to data breach notifications, in the event that data was held encrypted in a manner which prevented the attacker from gaining access to the underlying data, thus rendering it unreadable to them [16, Article 34, §3a].

Another potential avenue an adversary may explore to gain access to data include access gained through presumed-trusted computer systems, such as workstations used by staff, or publicly accessible (and tamperable) terminals. If the endpoint used to access secure data is not itself secure, any data accessible by a user of that system through any kind of credential entered to the endpoint is not secure. For example, a terminal in a retail outlet may be unsupervised, allowing a hardware keylogger to be installed. This would allow the user's credentials to be captured, thus granting the attacker access to any operation able to be carried out by the user. Alternatively, a piece of software like a Remote Access Trojan (RAT) could be used to gain remote access to the terminal, thus granting an unauthorised user the ability to access corporate systems using the legitimate and trusted terminal. The challenge of endpoint security is therefore relevant

[1] `https://www.spideroak.com`

to preventing unauthorised access to sensitive data in an organisation setting. It is also relevant for individuals wishing to ensure that their own systems are not subject to "evil-maid" style attacks [17], where their own trusted system is modified by a third party while the computer is out of sight. This could result in a keylogger or other piece of pervasive malicious software being installed on the system, perhaps as a replacement bootloader for their operating system, thus compromising the security of the operating system running on the computer, even where best-practice techniques such as full-disk encryption are used.

In an otherwise well-designed and secure service, one major avenue for unauthorised access is through social engineering. Indeed, this approach is widely recognised as being highly effective, and "bug bounty" programs typically make specific exclusions for social engineering attacks on staff [18, 19]. Given social engineering leverages privileged access of staff, any service not employing encryption in a manner which prevents access by the service provider is inherently vulnerable, to at least some extent. For example, scenarios where high-profile targets' emails have been accessed by third parties have highlighted the risks of social engineering being used against helpful staff, in order to gain access to accounts they should not have access, including, for example, the CIA Director's AOL email account [20].

1.1 Thesis Overview

This thesis explores the design of secure, decentralised network services and storage, and some of the challenges faced in the design and implementation of these services. It introduces general techniques, which may be used as a stepping stone towards a secure decentralised storage architecture, or which may themselves be applied more generally to existing architectures, to significantly increase the security of stored data. An analysis of an existing secure decentralised network, including demonstration of a number of novel attacks, is presented, as well as analysis of their severity and ease of realisation. A number of contributions are made towards the realisation of practical, secure information storage, whether on centralised or decentralised services, highlighting the importance of client-side security, and showing that current client-side security measures are typically inadequate. Widely used software, claiming to offer security through encryption, is shown to not be secure, with many examples found to make unsubstantiated claims as to their security. Finally, an exploration of identity within a decentralised network is carried out, given the differences between a centralised and decentralised service, and a novel means of handling identities securely on smartcards, without storage constraints, is presented.

The remainder of this chapter explains the motivation of this research, based upon the key problems encountered in secure storage solutions, and describes the main contributions of this work.

1.2 Motivation

This work is motivated by the challenge of allowing people to securely store their personal data, without fear of it being compromised, in order to preserve their right to privacy. While this does not inherently require the use of a decentralised network, security can be considered to encompass a variety of factors, including confidentiality and availability. In order to ensure the confidentiality of data, and thus offer users the ability to exercise their right to privacy, it is necessary either to have strict and effective access controls in place on the data, or to securely encrypt it, such that the data itself is not readable to an unauthorised user. Given the principle of defence-in-depth, there is clearly an advantage in doing both, therefore ensuring that even if access control were to be breached, the data retrieved would be rendered unreadable and unusable to the attacker. This is further motivated by the upcoming European General Data Protection Regulation (GDPR) [16]. Availability, on the other hand, encompasses the challenge of ensuring that a user is able to gain access to their own data at any time, without the failure of another party, be that technical or commercial, causing a user to lose access to their data. This is therefore another scenario where a decentralised means of storing and accessing data may be advantageous.

The key motivations of this work can therefore be summarised as follows:

- Affording individuals a strong assurance of the confidentiality, integrity and availability of their own data.

- Using these properties to offer users privacy choices which are enforced through cryptography rather than policy.

- Layering encryption on top of existing policies to offer stronger protection of data, to reduce the impact of breaches.

- Designing future systems such that user data and functionality is not lost in the event of a service provider failure.

The following sections shall provide some context as to the challenges faced by users of internet-based services in an era of increasing platform centralisation.

1.2.1 Limitations of Centralised Services

When centralised services are built upon other centralised services, a significant chain of potential failure may occur, whereby a provider of *platform-as-a-service* technology goes offline, as a result of their upstream database provider suffering from a prolonged fault [2]. Here, in addition to the customers of the platform provider finding their services are unavailable, many other services using the upstream provider are also made unavailable. For example, a database platform provided as a service, such as DynamoDB, may be used by many other products, such as Netflix, meaning that a DynamoDB outage could cause service disruption for Netflix users [21]. Decentralised services naturally remove this chain of dependency, and allow services to operate without failure of an external party causing a loss of service.

Additionally, in the event of a centralised service provider deciding to no longer provide a service, users of that service are often left with little or no recourse to allow them to continue using the service; since the service itself resides only with the service provider, and the user's data is typically held by that provider, users have no option but to stop using the service. With more centralised services, this increases the risk that users will find a service they rely upon one day disappears, or announces it will stop operating. At that point, users may be able to retventurebeatamazonrieve their data from the service, but will lose the functionality offered by the service. Decentralised service architectures allow users to be reassured that nobody can take away a service from them, and that it will continue to be available for them, irrespective of the actions of the developers. High-profile examples of this include Google Reader, the now-discontinued web-based RSS reader, and the Revolv Internet of Things hub, which was acquired by Nest and then closed [3], leaving customers with non-functional hardware.

In addition to the loss of availability of services, a number of high-profile data breaches have occurred in recent years, whereby unauthorised access was able to be made to personal data, on account of a centralised service provider's failure to authenticate users and enforce access control rules properly. This makes it difficult for a user to safely entrust a third party service provider with their confidential data, since a simple mistake or failing on the part of the provider may result in its exposure.

There are also clear benefits for user privacy when it is not possible for third parties to access user data without the user's consent. Many centralised services have built up business models whereby user data is regarded as an asset, and it is shared with others for the purpose of profiling and targeting users with advertisements. There are therefore benefits to individual privacy and transparency when it is not possible for a third party

[2] https://aws.amazon.com/message/5467D2/

[3] http://revolv.com/

to access a user's data without their express consent.

Therefore, to summarise, the following scenarios are envisaged as being desirable to mitigate, as well as practical to seek to solve through the use of encrypted decentralised services:

- Prevent data from being disclosed to unauthorised third parties via errors or weaknesses in service provider systems or procedures.
- Allow users to retain access to services and data, even in the event of popular cloud services experiencing outages.
- Prevent loss of functionality due to a service providing ceasing to offer their service.

1.2.2 Convenience versus Security

Centralised web services offer some advantages to users, such as the ability to carry out password resets in the event that a user forgets their password. Nonetheless, the risks of such "backdoors" in technology were clearly highlighted in the widely publicised case of Mat Honan. In under an hour, his "entire digital life was destroyed" — his Gmail and Google account was compromised and deleted, his Twitter account was taken over, and his Apple ID account was used to remotely lock and wipe his phone, tablet and laptop, causing the deletion of all of the data held on those devices [22]. The means through which Honan's accounts were compromised highlights the fundamental weakness of services where user data is accessible to any party other than the authorised user — social engineering attacks against companies' support staff, eager to be helpful, ultimately led to the compromise of the accounts and devices in question.

Based on Honan's account of events, it was possible for the attacker to gain access to his Amazon account using only his email address, name, and billing address, the latter of which were able to be looked up in a phone directory, and the former relatively widely available online. This was then used to gain access to existing details on his Amazon account, including the last four digits of the original credit card on the Amazon account, which was the security question for Apple's password reset system. This allowed the attacker to take control of the Apple iCloud account, thus gaining access to the associated email as well as the ability to block and wipe any computers, phones or tablets associated with the account [22].

There is therefore a clear risk to user data, where it is possible for a service provider to either gain access to a user's data, or to grant others access to a user's data. Users entrust service providers with their personal data, and have expectations of privacy, which are, as shown above, not always 8fulfilled. There is also no technical proof or guarantee that these privacy policies are enforced or followed by the service provider, if it is possible for them to gain access to user data or allow others access to a user's data, and deny having done so.

This example, while extreme, highlights the risks of data held in online services, and the ease with which even a technically-competent individual (a technology writer) could be attacked, causing significant disruption to their life, as well as significant data loss. This scenario highlights the extent to which current services and products are reliant almost entirely upon a security-by-trust model; where a user simply must trust third parties in order to protect them, but where these third parties fail to verify the legitimacy of those attempting to make use of this trust to re-gain access.

Had Honan used end-to-end encrypted services, it is likely that the attacks described above would not have been possible — services designed around a model where the provider was untrusted would have prevented social engineering attacks from yielding any useful result, and avoided unintentional disclosure of his information which later led to account compromise. Despite repeated reminders from many sources that users should stop using "facts" about themselves as passwords (such as family members' names, dates of birth, etc.), when access can be gained through account recovery processes as a result of such fact-based authentication, there is a clear disparity between the level of security expected and experienced. These recovery procedures themselves also frequently contain vulnerabilities [23].

There is, however, a clear trade-off between usability and security, at least within today's services. It is possible to design a very secure service which does not facilitate the recovery of an account following the loss of the password. This may be perceived as difficult to use by many users, particularly those who do not use password managers to store their credentials. The ability to reset a web service's password through an email link, while convenient, also reduces the security of the web service to that of the security of a user's email account. As users have been shown to generally be unable to select and remember secure, unique, high-security passwords [24], this clearly presents a usability challenge.

These challenges present opportunities for new approaches to the security of data, offering increased security and privacy of data, while also removing the availability risks of having a single point of failure in a service's architecture.

1.3 Towards Potential Solutions

There are several potential approaches to eliminating or reducing the risk of attacks such as those discussed above, and mitigating the impact of one succeeding. Not every approach will be appropriate in every scenario, nor will every attack be applicable in every situation.

One potential and signficant approach to reducing the impact and feasibility of security breaches is through the use and development of decentralised services, rather than centralised services where all user data is held under the control (and access) of one or a small number of companies. When data is not held under the control of one common entity, it becomes much harder to carry out attacks on a third party to gain access to another user's data. Had the CIA Director's emails been held in a decentralised manner, rather than under the full control of AOL, it would not have been possible for attackers to gain access to his emails by social engineering AOL staff using information obtained from Verizon [20]. While attacks against individuals remain possible in a decentralised system, there is no single point to attack, since all users' data is not conveniently residing on one entity's servers. This makes it more difficult to carry out a denial of service attack impacting a significant number of users, and means it is necessary to attack every individual server one-by-one, rather than there being a single large target to attack.

Despite this, in some situations it may not be realistic or practical to use decentralised services, as this thesis shall explore, particularly in the context of mobile access to services. For this reason, more general techniques to ensure the security of data held on a service are still relevant. Additionally, many of these can also be applied to decentralised systems, as the techniques themselves can be applied to various scenarios, in the design or implementation of services, or the security of client end-points for the purpose of protecting data from unauthorised access.

One fundamental difference between decentralised and centralised services is that typically, as a result of decentralised services holding user data on untrusted third parties' computers, all user data is encrypted by a key held only by the user. This is often referred to as client-side encryption, since the key is only accessible to the client, rather than the server. This approach offers a significant advantage for user privacy and security, since it is not possible for any service provider or third party to make available sensitive data — a user's data becomes as secure as they make it. Note that this type of encryption is not in any way unique to decentralised services; it is possible to create a client-side encrypted service in a regular centralised architecture. Client-side encrypted services, which protect user data against decryption by the server, remain relatively uncommon, yet are much more widespread in decentralised services as a result of the frequent desire to hold user data on untrusted computers. This allows for actual deployments and implementations of this technology to be studied. One challenge of client-side encrypted services is around the management of user keys for non-expert users — there is typically no process in place to facilitate the recovery of lost keys, meaning that users must take responsibility for securely storing their keys in a manner which will not be lost or compromised.

1.4 Key Research Questions

The thesis argument put forward here is that decentralised storage networks present significant opportunity for the protection of privacy and security of individuals' data, while also introducing the potential for new security considerations, as a result of their decentralised design. Therefore, the thesis will explore how security and privacy of data can be protected in such a network. The decentralised and distributed nature eliminates the concern of a single point of failure from systems. By building a secure storage network, other services and applications can then be built upon this secure storage layer. These points shall be explored through the following research questions:

- How vulnerable is a decentralised storage network to large numbers of malicious parties participating in the network, and are such attacks practical?
- How can a decentralised network retain security against large numbers of rogue users joining, without a centralised entity controlling access to the network?
- How can parties which do not trust each other reach secure agreements in a decentralised way, in order to allow decentralised services to be used as replacements for centralised platforms?
- Could mobile phones participate in a decentralised network in an efficient manner, given the typical requirement for persistent connectivity to participate in a decentralised network?
- To what extent are the devices and software used to access a decentralised network a risk to security, and how can these be avoided?
- If services were to be built upon a decentralised platform, how can users discover other users and content within that network, without relying on centralised discovery processes?
- How can a user protect and secure their own data within a decentralised network when, by definition, their information is exposed to other users?

- Can users feasibly and practically have multiple identities to allow for separation of different activities on a decentralised network, and reduce their exposure if some of their keys were to be compromised?

1.5 Thesis Outline and Contributions

Figure 1.1 illustrates the layout of the main chapters of this thesis, highlighting the relationship between chapters.

Chapter 2 presents an overview of security concepts used and referred to by other chapters, and explores some of the key principles of security, privacy, and the technology surrounding them, as well as some of the general rules of security and proper implementation.

Chapter 3 presents the first contribution of this thesis; a high-performance proof-of-storage system design, to prevent flooding attacks from gaining control of the network, by requiring and verifying that users contribute a useful resource to the network.

A thorough critical security evaluation of the MaidSafe decentralised network [25] architecture and implementation is also presented, to provide context for this contribution, highlighting significant performance gains which can be gained through a modified cryptographic implementation, without a loss in security. A number of threats to the network are introduced, with a novel implementation used to show that the uniform XOR metric cannot be assumed to offer security against determined attackers engaging in a Sybil attack, and results used to show a high-performance practical attack against a network relying on address uniformity for self-management. Finally, a decentralised protocol to allow for ownership-transferable data is contributed, facilitating the mutation and transfer of data on a decentralised network, in a manner which is verifiable to all other other network members.

Chapter 4 extends the work of Chapter 3 by considering solutions to some of the challenges introduced by decentralised services. It contributes a decentralised digital contracts solution, allowing for third-party verification of contracts, allowing users within a decentralised storage network to trade responsibilities to provide storage in the network, with parties providing storage able to charge for the provision, to offset the use of buyers. The protocol is threat-modelled, and an extended version is contributed, showing how both parties may create obligations upon the other, through the use of these verifiable contracts. A scheme for the implementation of decentralised services is also contributed, identifying the necessary high-level operations, and showing how these should be implemented to realise secure, user-facing functionality for decentralised services. This chapter also considers the feasibility of using mobile devices with decentralised network

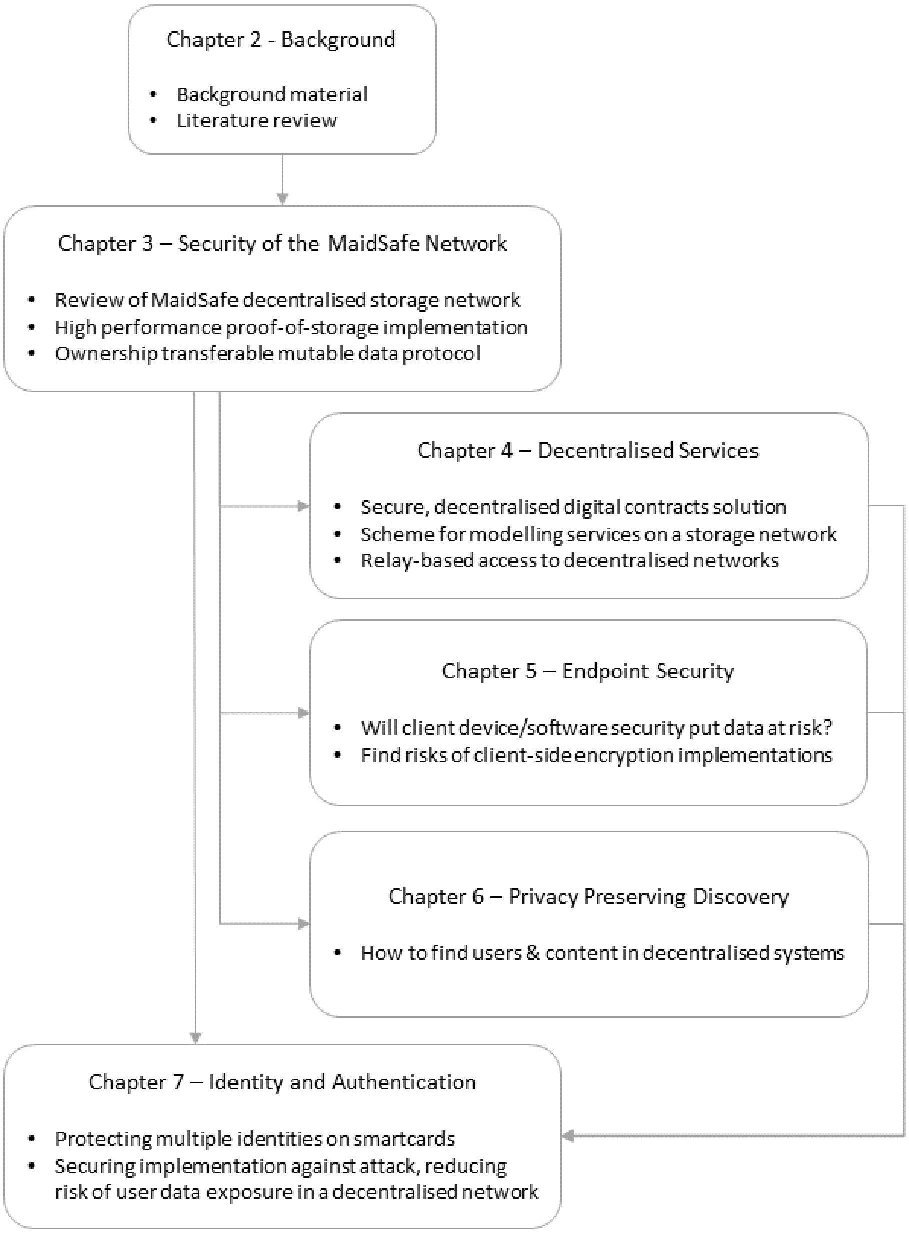

Chapter 2 - Background
Background material
Literature review
Chapter 3 – Security of the MaidSafe Network
Review of MaidSafe decentralised storage network
High performance proof-of-storage implementation
Ownership transferable mutable data protocol
Chapter 4 – Decentralised Services
Secure, decentralised digital contracts solution
Scheme for modelling services on a storage network
Relay-based access to decentralised networks
Chapter 5 – Endpoint Security
Will client device/software security put data at risk?
Find risks of client-side encryption implementations
Chapter 6 – Privacy Preserving Discovery
How to find users & content in decentralised systems
Chapter 7 – Identity and Authentication
Protecting multiple identities on smartcards
Securing implementation against attack, reducing risk of user data exposure in a decentralised network

Figure 1.1
Diagram illustrating layout of thesis and content of chapters

services, and contributes a hybrid relay approach, allowing users to join a decentralised network via a relatively untrusted relay.

The importance of the client software and devices used to access services is explored in Chapter 5. This chapter contributes a security evaluation of various software used on endpoint systems, to establish a view of the current state-of-the-art. Firstly, the widely-used Google Services in the Android operating system are explored, and it is shown that despite claims otherwise, certificate pinning was not in use, exposing user data to man-in-the-middle attacks if a certificate authority is compromised. It also shows that certificate pinning updates are implemented in an insecure manner, permitting the updates to be blocked from being installed. An architecture entirely trusting the server is shown to result in poor security decisions, such as sending all user keystrokes in the password field to a server over unpinned SSL. It was also shown that Google may remotely change the encryption password on devices using Android Device Manager, violating user assumptions of the security of device encryption. A review of third party encryption software is also presented, highlighting the lack of proper implementations of cryptography on a wide range of popular applications on the Play Store, claiming to use encryption to protect user data.

In order to make decentralised services useful, they should offer user-facing functionality to assist users with finding other service users, in order to permit them to interact. Chapter 6 contributes a novel solution to the challenge of privacy-preserving contact or user discovery on a third party service, without requiring the disclosure of information to the service provider which would identify a user, or allow them to form social graphs between users. A full analysis is provided of the level of security offered against attack, and the various attack scenarios are modelled based upon their difficulty and computational requirements. This offers a solution to contact discovery, a process which currently requires a centralised service to have access to information about all users of a service, and be trusted to act properly with this information.

Chapter 7 considers the challenge of establishing and authenticating users and their identities on services, which is necessary both for the security of a decentralised storage network, as well as for security of applications and services. The contribution of this chapter is a novel approach to storage of identities on smartcards, for centralised or decentralised services, allowing a single smartcard to be securely used with an unconstrained number of identities, thus permitting users to follow best practice and segregate their identities used on different services. A full security analysis is carried out for the implementation, showing the measures taken to prevent a variety of attacks, and the performance of the solution is demonstrated on a readily available and affordable smartcard.

Chapter 8 concludes this thesis, and introduces discussion about future work and challenges in the field of secure decentralised services. It re-visits the research questions posed in this introduction, and details how these have been answered.

Chapter 2

Background Material

This chapter shall review a number of background concepts used within the remainder of this thesis. Firstly, an overview of centralised and decentralised networks are given, as well as a history of decentralised network design and related work. Next, a more general overview of security is given, exploring the principles and a number of the technical constructs used in order to offer security. Finally, an overview of privacy and previous related work in the field is given, in light of the close links between security and privacy in the design of services.

2.1 Service Architectures

Within the context of this work, most services shall be considered to be either centralised or decentralised. Centralised services are typically those which are operated by a single provider, within their own infrastructure. This includes many popular web or cloud-based services. A common characteristic of centralised services is that the core functionality offered by the service would be lost by either a technical or business failure of the company behind the service. In contrast, decentralised services are designed such that functionality is preserved, even in the event of a failure of the technical or business side of the company behind a service.

2.1.1 Centralised Services

Many of today's internet-based services are built around centralised architectures. While, as shall be discussed below, the internet itself is fundamentally decentralised, it is possible for a centralised architecture to be used in the implementation of a particular service. An example of a typical centralised service architecture is shown in Figure 2.1, where a client device connects to a set of remote provider-operated services across a connection. All (or a large number of) service users communicate with these same servers.

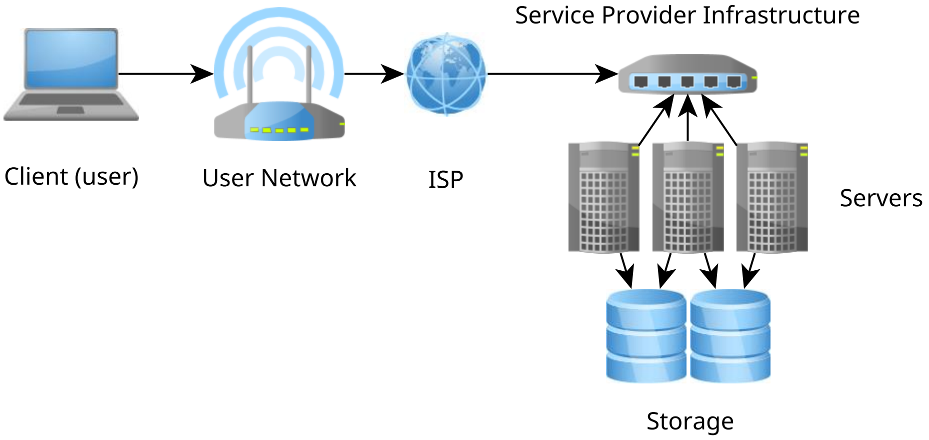


Figure 2.1
A typical centralised service architecture, where client devices communicate with a set of provider-controlled servers, holding their data under the control of this service provider and their servers.

While these servers may be distributed geographically throughout the world, users are still dependent upon one single entity; the provider of the service. If this company were to choose to cease providing the service, or cease trading, a user's ability to access the service would not be guaranteed. A similar outcome could be experienced in the event of a wide-spread cyber-attack resulting in the service being taken offline, for example.

This service is centralised, in that if the operator of the service ceases to provide the service, all utility of the service is lost by its end users. The failure of this single provider's architecture will result in the loss of availability of the service for all of its users.

In recent years, the expansion of cloud computing has resulted in a second layer of centralisation across internet-based services; the upstream providers of computing resources, storage and other services may well be heavily centralised, resulting in a waterfall effect in the event of an outage. When centralised services are built upon other centralised services, a significant chain of potential failure may occur, whereby a provider of technology goes offline, as a result of their upstream database provider suffering from a prolonged fault [26]. In this case, an Amazon DynamoDB cloud database experienced downtime. The knock-on consequence of this was the unavailability of a wide range of other products and services from Amazon's clients, including many significant and well-known websites [21]. In addition there was a knock-on impact on Heroku, a Platform-as-a-Service provider, which relied on Amazon cloud services for its underlying architecture, causing two significant outages for its users, of 8 and 7 hours respectively [27].

Here, in addition to the customers of the platform provider finding their services are unavailable, many other services using the upstream provider are also made unavailable.

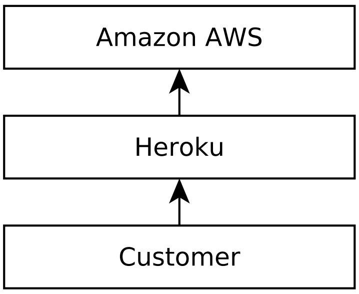


Figure 2.2
The chain of dependency of a Heroku-based service, with Heroku being the customer-facing platform provider, and Amazon AWS the underlying hardware operator.

The ability to remove this chain of dependency, and allow services to operate without failure of an external party causing a loss of service, is clearly an advantage for the resiliency and availability of online services. Such failures are not isolated incidents, with previous disruption causing major outages for major internet services [28].

2.1.2 Modern Web Services

Many modern internet-based services and applications are designed such that user data is stored by a single entity, and all users access their data from this entity. Today there are a number of highly popular web-based email services, such as Gmail, which remain a part of the federated email network, implementing the standard protocols, but which hold the email data of many millions of users under the control of a single company. In the event of a failure of their servers or infrastructure, massive numbers of people may be left without email, causing major disruption [29].

Similarly, when a user uploads data to a centralised storage service such as Dropbox, that data is entrusted by the user to Dropbox, who store this data on their own servers. When the user then wishes to access the data again, they authenticate themselves to Dropbox, and request the data be returned to them. In the meantime, this data resides on Dropbox's infrastructure, where the user must trust that Dropbox keep the data secure, and prevent others from accessing it. This trend towards centralised storage of information and service provision is relatively recent, compared to the history of computing and the internet. Early file and information retrieval protocols on the internet, such as gopher and FTP, provided similar functionality in a decentralised and federated way, as discussed in Section 2.1.4, which anyone could run without the user-friendly web-interface and product-based overlay provided by Dropbox.

In the context of service provision, a decentralised service can be considered one where users are not required to use a single provider's servers to receive access to a service. For example, email itself is a federated and decentralised service as you may install and operate your own server and client, but Gmail is a centralised mail service, since it is not possible to host your own installation of Gmail and hold your data independently.

It is also important to consider that there is a subtle distinction between the levels of decentralisation seen in services. For example, if one user's email server fails, they are unable to send and receive email through their server at that time. In contrast, Distributed Hash Table (DHT)-based decentralised services, which will be discussed later, remove this reliance upon even a user's own server, since there is no one-to-one mapping between users and servers; the user may connect to any available member of the network and gain access to their data.

User-facing functionality on the internet is increasingly centralised, as a result of Software-as-a-Service (SAAS) business models and products. The SaaS model for services has been described as where "the software system and users' data are stored off-site in a central location run by the vendor" [30]. From a perspective of security and privacy however, there are significant implications for users.

2.1.3 Decentralised Services

Historically and at its heart, the internet is fundamentally a decentralised network. Decisions on the routing of packets are taken at a local, per-router, level, using the Border Gateway Protocol (BGP) to identify available routes [31]. This means that in the event of disruption to part of the network, working nodes will continue to pass packets to their destination using alternative routes. This makes the internet inherently resilient, as there is no single entity which can prevent other entities from continuing to use the network, and inherent redundancy in connectivity between systems. In the event of the failure of a network link, packets can be routed to their destination using alternative routes.

This also applies to the low-level protocols used to provide connectivity and basic functionality on the internet. For example, HTTP and email are decentralised protocols, since users can operate and interact with their own, or others', HTTP or email servers. Indeed, the traditional services of the internet and subsequently world-wide web were almost always implemented in a decentralised manner, as a result of the inherently decentralised design of the internet, around multiple servers located in multiple locations, to which users may connect.

By virtue of this, it is possible to build services upon the existing internet infrastructure which are based on decentralised architectures. There are two common models for decentralised services on the internet; distributed and federated services. These shall now be considered.

2.1.4 Distributed and Federated Services

The Domain Name System (DNS) protocol is not a fully decentralised protocol, although it is designed in a distributed and federated manner. DNS allows for hierarchical naming and efficient look-up of computers on a wide-spread network, such as the internet. In order for such a system to work, however, it is necessary for users and server operators to agree upon a series of naming conventions, and a process through which names may be claimed on the system. This was implemented through the centralised process of DNS registries. DNS is distributed in that anyone may run a DNS server, and it will interact with DNS servers elsewhere on the internet to allow any domain name to be resolved. It is not, however, decentralised, since overall control of the root zones is vested in one entity (ICANN), which delegates authority to manage sub-divisions of the namespace to registrars [32].

Despite the centralisation of the DNS root, the DNS protocol itself makes it possible for anyone to host their own root zone, although in reality almost all networks advertise local DNS servers which use the standard Internet Corporation for Assigned Names and Numbers (ICANN) root zones. Open source projects have undertaken to provide alternative roots [1]. Many internet service providers and network operators provide their own DNS servers, in order to reduce query bandwidth and avoid excessive load on the root DNS servers.

Federated network services are those designed to use standardised communications protocols, facilitating interoperability and communication between users on different servers implementing the same standard. Users of different implementations or installations of a compatible service may interoperate and interact with each other, regardless of the internal structure of their own implementation of the service, provided their implementation adheres to the agreed-upon standards for interoperability between providers. Federation is seen in many early internet protocols, where users are identified based upon their username, as well as the hostname of the server they use. This can be seen today in email addresses — a user's email address is formed of a username, the @ symbol, then the hostname of their email server. This made email a federated service, where each user was able to run their own interoperable server, and send emails to anyone else with access to an email server. In the event of one email server failing, only those users either sending or receiving email from it would be affected. Other servers would operate according to the protocol, and attempt to redeliver mail when the recipient server was back online. Therefore, from a global perspective, email can be considered to be decentralised, since the loss of one provider will not prevent others from exchanging email, although for a given user, they are dependent upon their provider itself, which

[1] https://www.orsn.org/en/

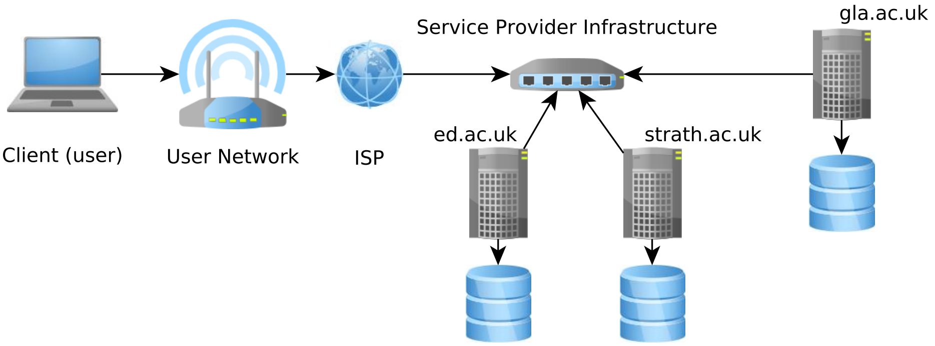


Figure 2.3
A typical federated service architecture

may be using a centralised architecture.

Figure 2.3 illustrates the architecture of a federated service, where multiple servers are operated by independent third parties, and a client may interoperate with these as required.

2.2 Impact of Centralisation on Security and Privacy

Centralisation, according to the Merriam Webster dictionary, is the process of bringing "something under the control of one authority", or "to bring things [...] together at a single point or place" [33]. Within the context of a network-based service, this could be described as a single company or entity having control of a service, and the data of its users. Note that, with the advent of cloud computing and outsourcing of storage, the structure and operation of the service itself may be centralised, even though data may not necessarily be geographically centralised [34]. For example, the popular Dropbox cloud storage service uses Amazon's S3 cloud storage product to store user data [35].

The original intention of the internet when it was first designed was for research, rather than for commercial use, and as such, security was based upon mutual trust, respect, and assumed honour and good intentions of others with access to systems [36]. Indeed, due to the cost of computers at the time, it was simply accepted that they would be shared. Therefore, it was acceptable to store email on a shared email server, since it was assumed that those with administrator access to the system would not read or interfere with their emails. These previously-unwritten rules were formalised by the Usenix Special Interest Group for Sysadmins, in the *System Administrators' Code of Ethics*, which states "I will access private information on computer systems only when it is necessary in the course of my technical duties. I will maintain and protect the confidentiality of any information to which I may have access, regardless of the method by which I came into knowledge of it." [37].

More recently, however, the cost of computers and internet connectivity has reduced incredibly — free wireless access to the internet is now available to the public in towns and cities [38], and internet-capable devices are available for as little as £10; £5 for a Raspberry Pi Zero, and £5 for a WiFi adapter. Despite one computer per person now being practical, and indeed in many cases more than one computer being the norm [39], today's internet services continue to be built with the assumption that users should provide all their data to a service provider, which holds this data on an external computer, and is trusted to hold the user's data securely. While this offers some elements of convenience, since users may easily access their data from anywhere that this server is available, it has also resulted in the effective centralisation of significant quantities of important, and sensitive, user data.

In contrast to the strict historical expectations of systems administrators keeping all user data private, and only accessing it for technical duties, today's service providers no longer uphold this principle.

For example, Google's Gmail service is widely known to read user email, for the purpose of building advertising profiles of users, which may be incorporated into its AdSense product, which shows advertising across the wider internet, based upon the content of a user's email inbox [40]. Similarly, Yahoo mail defaults to scanning user email for targeted advertising [41]. Despite widespread criticism [42], Microsoft has asserted that it is allowed to choose to carry out "content pulls" of user accounts based upon internal legal review, due to its terms and conditions, and that it may look through these for their own purpose - CNN described how investigators "pored over emails and instant messages", since "the servers storing the information are on its own property" [43].

Indeed, The Guardian has highlighted that most webmail services claim rights to read user data for their own reasons to protect themselves [44]. Such terms and conditions are instituted as contracts between users and the service provider, and therefore must be accepted for users to make use of the service.

This highlights a fundamental limitation in the threat models used when designing such products and services — it is clear now that service providers do not view themselves as posing a threat to user security or privacy, and therefore consider themselves as a trusted entity within the design of the service, allowing themselves to authenticate and verify users, and carry out other similar tasks. This assumption has been questioned by Berners-Lee, original developer of the world wide web [45], as well as by others [46], as the inherent centralisation of data is creating both dependency upon a small number of companies, as well as opening up the actions of users to this small number of companies.

In contrast, privacy-preserving and high security alternatives would consider as many components of a network and service architecture as possible to be untrusted, with trust only placed in the absolute minimum of components necessary to ensure operation of the service. The assumption that the service provider is trustworthy is, as discussed previously, now known to be misguided, given that service providers have now shown on multiple occasions that they do not regard user data as confidential and inviolable. In any case, the risk of a security breach yielding access to vast quantities of user data has only further highlighted the risks to users when they must entrust third parties with their data in order for access to a service.

2.3 Distributed Hash Tables

One of the major challenges within a decentralised network is the efficient location and retrieval of data, with much previous work having focused on addressing this [47, 48, 49]. The challenge is somewhat different on centralised systems, where inter-node latency is typically considerably lower, as servers working together may be located within the same rack of a server cabinet. In contrast, peers in a decentralised network may be located at opposite sides of the world, posing challenges for data discovery and routing of requests.

In a conventional network, data is, by default, only held on that one system. In the event of its failure, the data would be lost if there was no backup copy available to be restored. This principle underpins the fundamental operation of individual computers, and means that even within a decentralised and federated system, there is still a risk of service unavailability or data loss, in the event of a system failure. As a result, various techniques for the storage of multiple copies of data exist. An innovative design of interconnected network, referred to as a Distributed Hash Table (DHT), makes it possible to overcome this, while remaining entirely decentralised. Within a DHT, many computers join the network, each taking an address within an address space, derived from the output of a cryptographic hash function. A more detailed explanation of cryptographic hash functions is given in Section 2.4. The uniform output property of this hash function results in a probabilistically uniform distribution of nodes within the address space of the DHT. Also within this address space are pieces of data, which can be considered as chunks. Each chunk has an address, and the data is stored by the network members located closest to the data's address. Note that this is closeness based on logical addressing, based solely upon DHT addresses, and not based upon physical proximity. A variety of DHTs were presented in parallel in the literature, including Chord [50], CAN [51] and Kademlia [52].

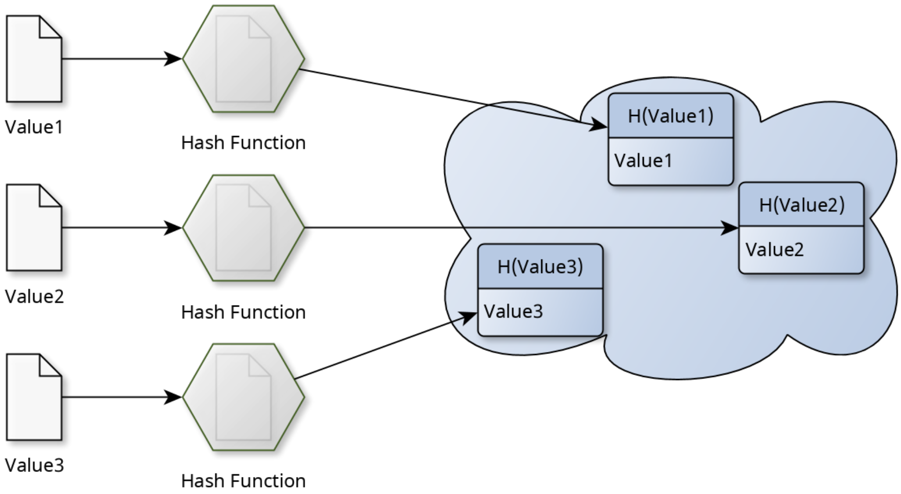


Figure 2.4
Use of cryptographic hash functions within a DHT for addressing

Figure 2.4 illustrates how cryptographic hash functions may be used to determine the address of a given piece of data. The DHT can thus be effectively considered as a key-value store, with the key being the network address, and the value being the data itself, such that the key is equal to $H(value)$. This is the concept of content-addressable storage, as proposed by CAN [51].

Multiple nodes with an address close to a given piece of data may be responsible for the storage of that data, meaning that the loss of any one of these nodes does not prevent the data from being accessed. The Kademlia DHT network concept has been extended and widely deployed as the mainline Bittorrent DHT, which was estimated to have between 15 and 27 million daily users in 2013 [53].

Within the Kademlia DHT [52], each member forms a routing table of N entries, for a network with nodes using N-bit addresses. By calculating the XOR of two network addresses, a closeness metric is obtained between the two nodes. Each routing table entry is considered as a *bucket*, containing a number of nodes' information. Working through the routing table, the buckets found should contain routing information (IP and port) for nodes which are of a given distance from the node in question. For example, the first routing bucket will contain details of nodes which share zero bits in common with the seeking node's address. The second routing bucket would contain details of nodes with the same first bit of their address. The third bucket would contain details of nodes sharing the first two bits of their address with the seeking node. This continues for each bit of the address, meaning that the probability of finding members of the network to fill the full routing table will decrease for later routing table records [52].

By way of example, Figure 2.5 illustrates this routing process in a simplified two-step

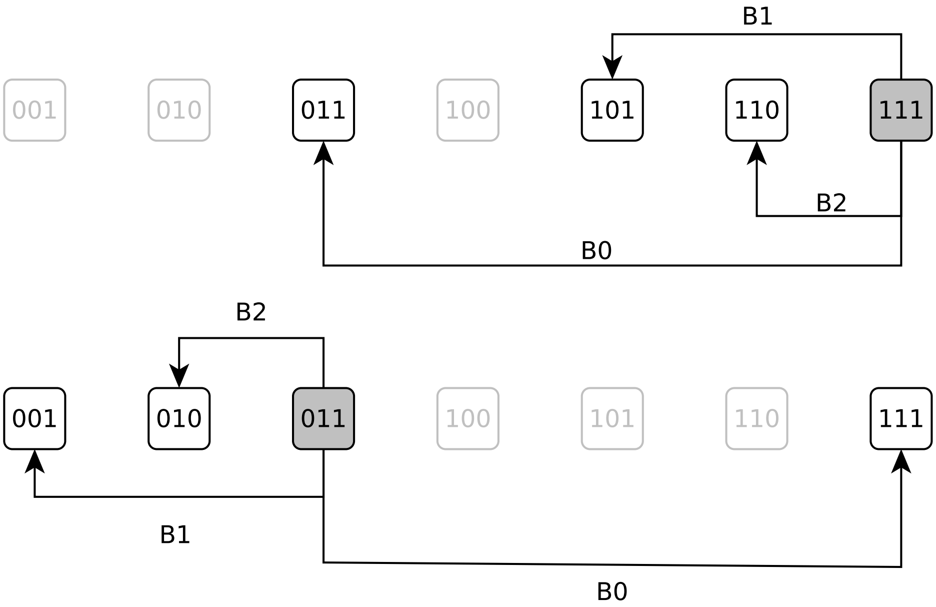


Figure 2.5
Illustration of XOR-based DHT routing, showing the routing process from the perspective of node `111` attempting to locate node `001`. The annotations on links between nodes indicate the respective bucket which a target would sit in, based upon address prefix. Node `111` can locate node `011` in its *zero* bucket (due to sharing a prefix of zero bits of the address). and this is the closest known node for node `111`. Node `011` can locate node `001` in its first bucket (due to sharing a prefix of one byte), thus allowing message delivery.

scenario, where node `111` wishes to communicate with node `001`. In this scenario, each node has a bucket size of 1, meaning that for each bitwise XOR address prefix, information on only 1 node will be held. In the first step, node `111` attempts to locate node `001`, and is unable to do so from its own routing table, since not all nodes are visibile to it (and those hidden nodes are made lighter in Figure 2.5). Since it is unable to locate the exact node, node `111` contacts the closest known node to the target node. This is node `011`, which is referenced in the bucket for having zero common prefix bits with node `111`. The lower half of Figure 2.5 then shows the operations carried out by node `011` — it is able to locate node `001` in its bucket for nodes sharing the first common address bit. Node `111` can thus communicate with node `001` by communicating messages to the closest known node to the address, which will either know the node, or know other closer neighbours of the node.

The end result is that each member of a Kademlia network forms its own routing table based on its view of the network. Within this table it identifies groups of nodes at approximate given distances from itself. By virtue of the bitwise addressing, a node will know more peers whose addresses are closer to itself than those which are further away. This minimises the size of the routing table which is required on each node, making the network addressing system scale. To locate another node within the network, the client node should attempt to contact the closest available node it knows of within the network. That node will have a more precise knowledge of nodes addressed closely to it, and can therefore assist with locating the desired node.

When data is stored on the network, it is held by the nodes with addresses closest to the address of the data, meaning that there is inbuilt redundancy and resiliency against loss of individual systems. Since messages can be routed to any other node in the network using the recursive routing process described above, it is possible to join the network at any location and gain access to data from anywhere, eliminating the impact of nodes joining and leaving the network during normal activities.

By placing computers within such a network, based on their logical DHT address, in addition to data, it is possible to create a distributed, decentralised data store, as demonstrated by Kademlia. The MaidSafe network, the subject of security analysis within this thesis, is based upon the Kademlia network concept.

2.3.1 Previous Decentralised Networks

A number of previous works have introduced fully decentralised network services for storage. Freenet was introduced in 2001, designed to offer full decentralisation of all network functionality, as well as anonymity for those sharing and retrieving content, as well as providing deniability for systems holding content within the network, such that they had no knowledge of the data being made available by their system [54]. One major limitation of Freenet, however, is that it is not intended for the permanent storage of data, and that which is not retrieved is purged from the network to make space for new data on the network [54].

The GNUnet project [55] was presentd in 2002 as an approach towards an anonymous and distributed backup system for files. It was later extended by the authors to include support for a censorship-resistant protocol for distributed file sharing, where content can be addressed by unique friendly names, in addition to key-based identifiers [56]. Previous work has identified a number weaknesses within the anonymity offered to users, as well as the ability for content filtering or censorship to take place on the platform, as a result of the design of the system [57].

2.4 Cryptographic Hash Functions

Cryptographic hash functions, which are pivotal to the understanding of DHTs, are designed with several high-level properties necessary for the design of secure systems. The four main properties of a cryptographic hash function are that [58, Section 5.3.1]:

- It is non-invertible, so given the output of the function, the input cannot feasibly be determined
- The hash function can be computed for any input, in a computationally efficient manner
- Any alteration to the input will change the output of the function significantly
- It should be infeasible to find two non-equal inputs to the function yielding the same output

The one-way nature of the function, its non-invertibility, is often referred to as pre-image resistance. A pre-image attack involves attempting to find an input to the function to produce a specific output. If it were possible to find an input to produce a given output, this would clearly violate the one-way nature of the hash function, since it would allow the "reversal" of an arbitrary hash output.

The differing output for differing inputs can be considered as second pre-image resistance, such that for a given input (x), it is infeasible to determine another input value y, such that $H(x) = H(y)$. Finally, as an extension of this propery, collision resistance covers the property of ensuring that it should be infeasible to find any two input messages, such that $H(x) = H(y)$. Note that for the property of collision resistance, this requires resistance to the birthday paradox scenario, as explored in more detail in Section 3.4. Since for the scenario of measuring collision resistance, both input messages can be selected, it is necessary for a hash double the length of that required to protect against a second pre-image attack [59, Section 9.7.1][58, Section 5.3.1.2].

From the perspective of designing a secure system, the properties of cryptographic hashes provide a number of useful constructs. Firstly, a cryptographic hash is usually assumed to approximate a uniformly random variable [59, Section 9.7.1]. This means that, for a series of arbitrary inputs, the output of the function should provide a uniform distribution across its full range. Coupled with the fact that a trivial alteration to the input should significantly change the output of the function, through its uniforimity, a hash function serves well as a checksum, and also as an authenticator in a length-constrained cryptographic construct, such as a block-based cipher or signature. For this reason, cryptographic hashes are widely used in the generation of digital signatures, where mesasge undergoing an RSA signature should be shorter in length than than the public key.

2.5 The MaidSafe Network

The MaidSafe network [2] is a proposed decentralised secure storage network architecture, inspired by the concept of the Kademlia DHT. Within the MaidSafe network, user data is stored across a large, peer-to-peer network, as described by above. To preserve confidentiality, data is stored in an encrypted manner, such that it is not readable by those holding it, and files are split into chunks, with these chunks distributed throughout the network [25]. Crucial to the concept of MaidSafe is that no one entity, not even the developers themselves, is capable of exerting centralised control over the network [60]. Decisions must therefore be made in a decentralised and distributed manner, and the network rules must be designed and carefully balanced to ensure that, in the absence of a centralised gatekeeper to ensure everyone "plays by the rules", parties will not be able to create disruption through refusing to play by the rules [60]. The MaidSafe network is an interesting target for research, since prior works have tended to focus on the transient transfer of information through decentralised networks, rather than on persistent storage of data for longer periods of time.

The MaidSafe network is built around a security model whereby no network entity other than the owner of data, or their authorised delegate, may decrypt it. Content is therefore transferred and stored by parties who are incapable of decrypting it [25]. In order to create a secure network with these properties, while ensuring data integrity and availability is preserved, a robust data storage strategy is needed. The approach taken by MaidSafe to achieve this is now explored.

2.5.1 Data Storage and Encryption

Within the MaidSafe network, data is always stored in an encrypted form. To allow for global deduplication of data, convergent encryption is used [61]. Convergent encryption [62], proposed by Douceur *et al.*, is an implementation of encryption, whereby the key used to encrypt a file is derived from its plaintext contents. Therefore, since the key is derived in a one-to-one mapping from the file contents, different users holding the same file will each produce the same ciphertext, allowing for inherent deduplication if this can be detected, thus avoiding the need for multiple copies of the same data to be stored [62]. This deduplication is achieved within the MaidSafe network, since data is addressed based upon the cryptographic hash of its encrypted contents.

An illustration of this scheme is shown in Figure 2.6 — the client splits their file firstly into chunks, then derives a key from the cryptographic hash of the chunk contents (represented by the yellow one-way arrow function). This key is then used to encrypt the

[2] http://www.maidsafe.net

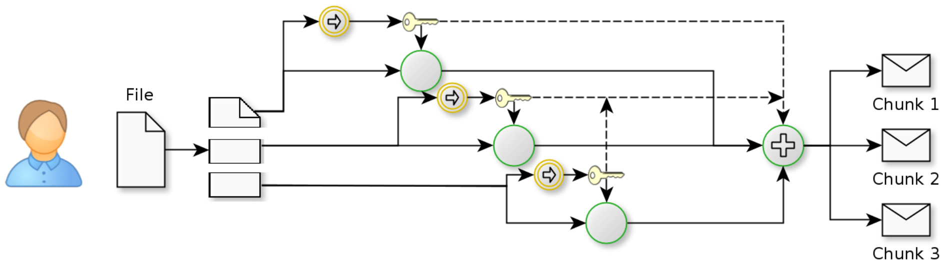


Figure 2.6
Illustration of MaidSafe self-encryption process, showing the chunking of a file, followed by derivation of a key for each chunk, then the encryption of each chunk, and finally the combination of chunks for XOR-based obfuscation.

chunk, and the keys are later combined to obfuscate (using the XOR function) the output of the cipher outputs, prior to yielding chunks ready for storage within the network. A more detailed analysis of the actual implementation of this process is presented as a contribution in Section 3.2.

The MaidSafe storage model incorporates three entities — the client, the chunk information holder, and the chunk holders. Both the chunk information holder and the chunk holders are vaults within the network. Each chunk is distributed redundantly throughout the network 4 times, thus requiring a layer of decentralised management to ensure that these copies remain, even as nodes join and leave the network. A chunk information holder therefore is a record which acts as a pointer towards the actual copies of the chunk data. This is illustrated in Figure 2.7. A user wishing to locate a chunk will make a query for data under the hash of the chunk's contents (its address). At this address, and stored by the 4 closest nodes on the network, will be a chunk information record, containing information about the locations where the actual data is held on the network. The chunk information record acts therefore as a set of pointers, held and maintained by the closest 4 nodes to the chunk's hash. This record supplies the addresses at which a chunk itself is currently available on the network. The client may then retrieve this data from the nodes holding the chunks.

2.5.2 Data Retrieval and Bootstrapping

Since data within the MaidSafe network, and in wider DHT-based services such as BEP-0044 within Bittorrent [63], is held at unpredictable addresses within the network, based upon the output of a cryptographic hash function, there is a challenge in locating data. In order to retrieve data, it is necessary for a user to store both the DHT addresses of each encrypted chunk, as well as the original cryptographic hash of each plaintext chunk. This is a result of the decryption key being derived from the plaintext hashes of

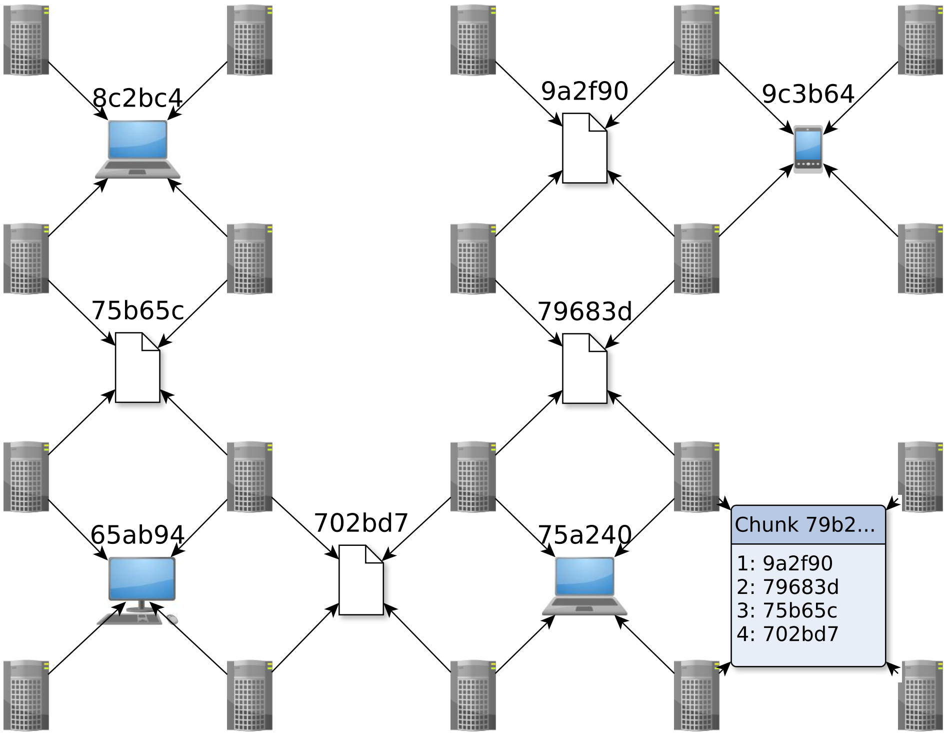


Figure 2.7
Illustration of MaidSafe chunk information holder (lower right), holding records of the addresses of chunks, which are distributed throughout the network. Chunks are logical, and are held by their 4 closest surrounding nodes, according to the XOR of the node address with the chunk address, as illustrated.

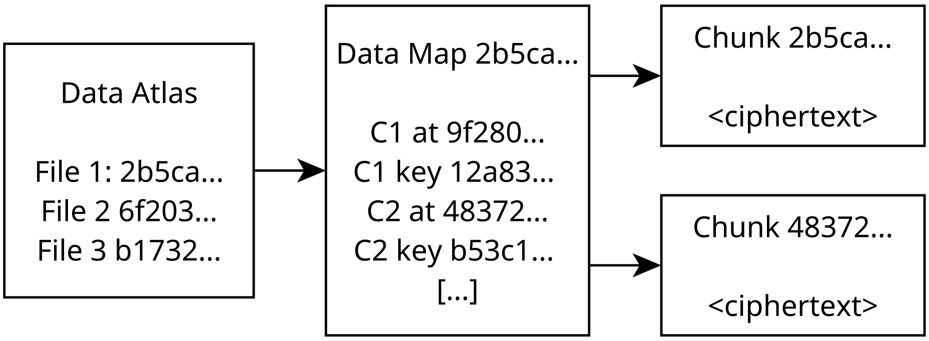


Figure 2.8
Illustration of the MaidSafe process for retrieval of a chunk, by first locating the account's data atlas, and using the listing of files to locate the data map. This data map contains the address of the ciphertext chunk on the network, and the key needed to decrypt it.

chunks, and the chunks being held at addresses based on their encrypted contents, as discussed in Section 2.5.1. In order to achieve this, the MaidSafe network introduced a concept of a data map, which acts as a mapping between plaintext data chunks and the respective ciphertext addresses in the DHT [61].

Each file would therefore have a data map, acting as a pointer to the chunks within the file, and record of the keys needed to decrypt the data. The process of locating these data maps is the next significant challenge for storage of data within a DHT — since the data map contents is not predictable, it must be stored in order to re-gain access to data from the network. In the absence of the data map, it is not possible to locate or decrypt the data. This also means that, in order to access data from a new device in future, without any a-priori knowledge of the data map contents, the maps themselves must be stored on the network, leading to the natural question of how to locate those maps without other knowledge.

The solution used by MaidSafe is the creation of a data atlas, a fixed-location index store of mutable data, placed at a location selected by the user, containing the keys and addresses needed to locate and decrypt the data maps and thus re-gain access to a user's data. The data atlas is stored encrypted at an address, with both the decrytion key and the address derived from a user's account credentials, thus meaning that the only information necessary to gain access to the network is knowledge of account credentials. Figure 2.8 illustrates the process through which a user may retrieve their data from the network, through the use of a data atlas and data map.

2.5.3 MaidSafe Account Handling

In order to protect a user's data from being modified by other users, and create persistent identitifiers within the MaidSafe network for trust and reputation, it is necessary for an authentication system to permit a user to identify themselves on the network, such that the identity of a user making a request can be proved to other nodes if required, for the purpose of the reputation system. Other nodes on the network may then use access controls, based upon this identity and authentication, to control which actions they will accept from a given node. It is important to note that there is no central identity or authentication server, and that authentication takes place with other parties a user node interacts with on the network. For example, a storage node may reject management instructions from a user's node which has not identified itself as a manager, and proven their identity to be one of their managers based upon their public key.

In order to ensure good usability of the network, the MaidSafe network has been designed such that a user should be able to join the network from a new device, without the need for them to have any a-priori knowledge of secrets or keys. Therefore, it is important to ensure that the login process is secure against other parties involved in the authentication process, including other nodes authenticating a node, since any actor able to uncover a user's identity or keys within the network would be able to assume their identity.

The MaidSafe model of self authentication and login is described in [64]. The premise is that a user enters a set of credentials, incorporating a username and password. The username and password are combined in a deterministic process involving a cryptographic hashing function, to derive an account salt. A randomly generated value is then defined as the "account access packet". It is symmetrically encrypted using a key derived from PBKDF2, incorporating the username and password of the account. The resulting encrypted data is then stored to the MaidSafe network as encrypted data, at the address of the hash of the username combined with the account salt.

Finally, a data atlas is generated and stored at an address generated by combining the username, password and account access packet. The data atlas is encrypted using a key derived through PBKDF2, incorporating the account's random string, as well as the account password. The encrypted data held within the account access packet is the "data atlas", as referred to in Section 2.5.2.

Figure 2.9 illustrates the process of derivation of the account login credentials and keys, according to the original MaidSafe paper on the concept [64]. Note that for security, the hash function used to concatenate the username and password to form the salt should be a slow hash, in order to prevent efficient brute force attacks against usernames and passwords.

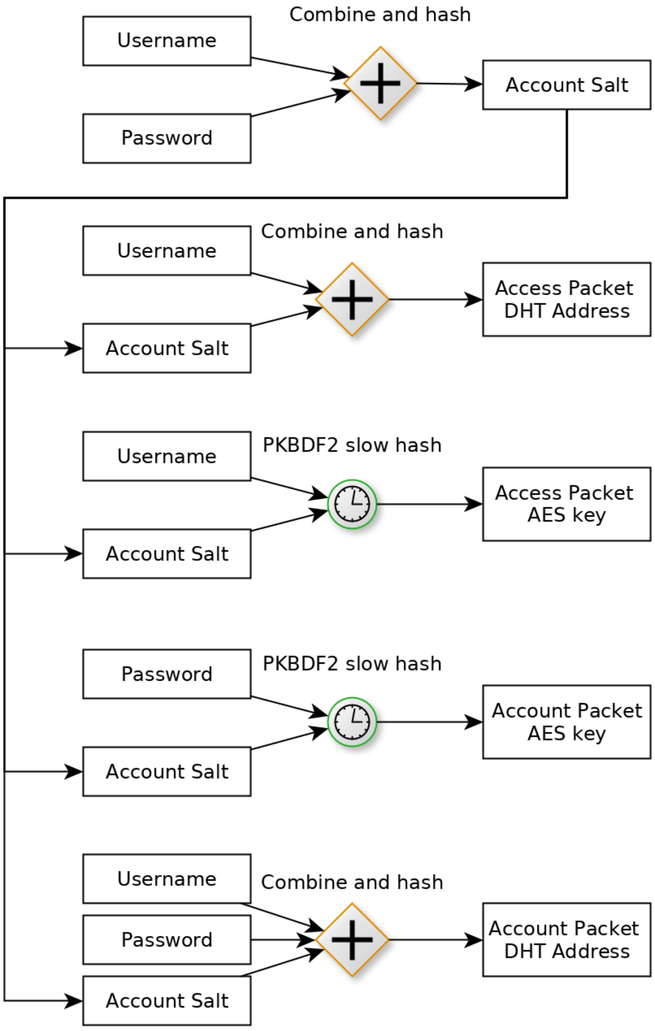


Figure 2.9
Illustration of the MaidSafe self authentication process, the derivation of account salt from username and password, and the use of the username, password and salt for the generation of keys and DHT addresses. This process is described by MaidSafe in [64]

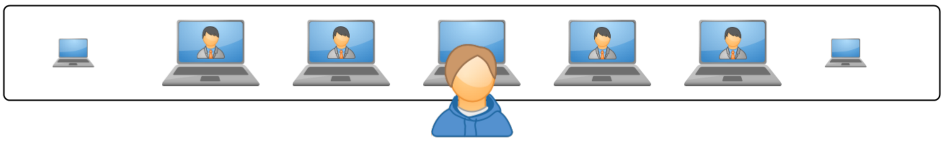


Figure 2.10
The client in the centre of this figure is managed by its 4 closest nodes

2.5.4 Network Management

To ensure the truly decentralised nature of the network, and avoid the need for any centralised coordination or management of nodes on the network, the MaidSafe network is designed to be self-managing. Under the concept proposed by MaidSafe, vaults are responsible for the management of not just other nodes, but also for assigned blocks of data. When a chunk of data is stored in the network, its data manager is set as the closest (by address) vault. This vault is responsible for distributing 4 copies of the chunk to other vaults, and then ensuring these vaults store this data and keep it available. When a client seeks this data, they simply request information on its location from the data manager, which will inform the client of some current locations of the data. As nodes go offline, this data manager will create additional copies of the data on more vaults, such that there are always sufficient copies available. Figure 2.10 shows how the user's node is managed by its 4 adjacent nodes in the distributed hash table (by address).

It is worth noting that due to the uniformity of the SHA-512 hash algorithm, as used in MaidSafe, the closeness of two addresses is not related to their geographic closeness — indeed, the addresses will statistically be geographically uniformly distributed, compared to the geographical distribution of MaidSafe users.

The security of relying on uniformity of hashes within this self-management construct has however not been reviewed, and therefore one of the main contributions of this thesis is a substantive critical review of the self-management protocol, along with identification and implementation of a number of novel attacks, as well as a review of mitigation strategies.

2.5.5 Security Model and Encryption

Data is secured on the MaidSafe network following the basic premise that the user's client device is not compromised, but that any and all other nodes on the network may conspire against the user. Data confidentiality should always be preserved, even in the presence of an entirely untrusted network. The network is designed to make use of a reputation system, which ensures that in a predominantly honest network, the majority of users will prevent dishonest network users from affecting the usage of honest users.

This is intended to prevent denial of service, since any node which fails to follow the network rules would no longer be considered reliable, and thus would not be relied upon to properly hold data [60]. It is presumed that users retain knowledge of their own username, password and PIN (all of which are used to generate a unique per-user key, which finds and decrypts their own account data, after retrieving it from the network) [64]. The MaidSafe login process was explored in more detail in Section 2.5.3.

Security on the MaidSafe network is multifaceted, and is not based upon any single assumption (aside from the client device not being compromised). Data confidentiality is preserved through the use of symmetric AES encryption on all chunks, with the key derived from the contents of the data, as described in Section 2.5.1, thus forming convergent encryption, as introduced in [65].

2.5.6 Deduplication

According to analysis by IDC, only 25% of digital data stored by users is unique [66]. This means that the storage of user data is inherently an inefficient process, involving significant duplication. The process of deduplication ensures that multiple copies of the same data are not needlessly stored, by recognising when a file (or constituent blocks of a file) already exists on the storage system. The president of industrial analysts DCIG, is quoted as saying that in a corporate environment, deduplication can allow a company to store 20x more data with the same storage faclities [67]. In an analysis of backup storage, Meyer and Bolosky found that across four weekly backups, file-level deduplication achieved a saving of 72% in storage requirements [68]. While clearly not all users will want to make use of a decentralised storage network solely to store full backup images of their computers, the ability to deduplicate data globally at file-level through decentralised platforms such as MaidSafe appears likely to be able to offer significant reductions in the storage capacity necessary on the network to store files.

Conventionally, encrypted data is not easily deduplicated between different users (since the same file would exist differently for each user, depending on their encryption key). Within the MaidSafe network model, data is globally deduplicated, while still being encrypted, by deterministically deriving the encryption parameters (key and IV) from the file itself. As highlighted in previous works, convergent encryption allows for a known-plaintext attack, in that any party with access to plaintext data may derive the encrypted version which will be stored on the network [69]. Various techniques exist which allow end users to mitigate this, such as using a per-account *salt* to allow for deduplication within data held in their own account, without exposing their data to known plaintext attacks by third parties [3], although these prevent global, network-level dedu-

[3] https://www.mail-archive.com/cryptography@metzdowd.com/msg08949.html

plication from taking place. The MaidSafe network has therefore elected to use global deduplication despite these risks.

By using this global file-level deduplication, data remains encrypted (and thus confidential), while only requiring the space for a single copy on the network. In order to provide reliable operation and resiliency against nodes going offline, data on the MaidSafe network is replicated 4 times to randomly selected areas of the network. As a result, given only a quarter of user data is unique [66], this balances with the replication factor of 4, to result in approximately the same storage requirement as that of data held by users.

2.6 Key Challenges in Decentralised Networks

In the absence of any centralised control over a network such as the MaidSafe network, there are a number of challenges introduced, which would conventionally be easily mitigated through the use of a trusted entity in a centralised architecture. Within a decentralised network, lacking such a trusted entity, these challenges pose a greater consideration, which this thesis shall address.

2.6.1 Resource Management in Decentralised Networks

Since the very early days of peer-to-peer decentralised systems, resource management has been an important consideration, to prevent a small number of users from taking advantage of the generosity of others [70]. This was illustrated by research conducted by Sen and Wang on the FastTrack network, which showed that only 1% of network users were providing 73% of the network bandwidth needed to share files [71]. In an earlier study carried out on the Gnutella peer-to-peer network, Adar et al. showed that 70% of users were not sharing any files, and that only 1% of all users were actually providing responses to half of network requests [72].

While these previous works have focused on the use of decentralised networks for file sharing, the challenge identified remains the same — if a network lacks adequate precautions to protect against free-riding, there is no individual incentive for a user to contribute to the network, be that by uploading data for others, in the context of a file sharing network, or holding data for others in a storage network. Since it has been demonstrated that significant quanitites of users are willing to free-load [72], it is clear that for a decentralised storage network to succeed, it must be balanced to ensure users may store data, but cannot use excessive quantities of storage to the detriment of others.

2.6.2 Application of Storage Limitations

Conventional, centralised, storage services often use a quota-based system of resource control, where users are permitted to store up to a certain capacity of data [73]. Beyond this point, the user would either not be able to store further data, or may be required to pay the service provider for an increased storage quota. Within a decentralised network, this is not necessarily possible without trusting a single entity to accurately and fairly enforce these quotas, and identifying which users should receive payment for storage is non-trivial.

It is therefore clear that it is necessary to have a decentralised means of restricting storage utilisation. Given the MaidSafe network has no central entity controlling account creation, it is not immediately apparent if it is possible to offer users a free storage quota, since such quotas could be defeated through creation of multiple accounts. A means of restricting storage available to users, without requiring a centralised trusted authority is therefore desireable. Ideally, it should allow for the variation in storage quota based on contribution, be that financial or by provision of storage itself, in order to be flexible and allow users to contribute to the network, and gain more storage allowance.

In previous works, storage quotas have typically been centralised in operation. For example, in the work of Druschel and Rowstron, storage quotas depended upon centrally issued and trusted smartcards, which contained a user's storage quota [74]. While users could offer more storage (in exchange for an increased storage quota), and audits over provided and used storage were carried out, the network ultimately depended upon a smartcard issued by a trusted centralised entity, as "enforcing quotas in the absence of a trusted third party would likely require complex agreement protocols" [74]. Through the decentralised and peer-managed approach of the MaidSafe network, it may be possible to achieve this without explicitly trusting any individual authority, since all statements are verifiable by any other user of the network [60].

The most obvious means of enforcing storage limitations on users, in a decentralised manner, is to allow users to store a quantity of data proportional to the quantity of data they themselves are storing for other users, as was originally proposed by MaidSafe in discussions during this research. This ratio could be one-for-one, allowing users to store the same amount of data as they hold, or it could be different, allowing users to store more or less data than they themselves store for other users, based on other factors. One example would be base storage upon only unique data, since network deduplication within the MaidSafe network, as discussed in Section 2.5.6, would ensure such data was not stored again. Another approach would be to share the cost of storing a given piece of data which multiple users wished to store between all users also holding that data. Therefore, users holding non-unique data would only be charged proportionately to the number of existing holders of the data.

The work of Ngan, Wallach and Druschel presented the concept of auditable and decentralised quota management, however required an auditing process which required a public ledger to be permanently available, detailing a list of the identifiers of every file being held by a vault, and similarly, a list of the identifier of every file held by a client [75]. While this is a significant improvement upon the requirement for a trusted centralised entity in the original proposal by Druschel & Rowstron [74], the presence of such public audit logs poses a risk to the privacy of users on the network — users storing specific data could be targeted, and vaults providing their storage could also be identified and targeted.

While MaidSafe have proposed the concept of offering free storage to users, without them being required to contribute resources, up to the average storage utilisation per user, this thesis argues this is too easily abused, given there is no way to restrict how many accounts one user can create, i.e. a "Sybil attack" [76], where many accounts are created by an attacker, and joined to the network for the purpose of gaining a majority in voting protocols. Indeed, it would be quite conceivable to make a modified fork of the open source MaidSafe client, which could unify multiple accounts, allowing users to effectively gain unlimited storage without contributing resources to the network. Naturally, this would be of significant detriment to the network. While means of restricting account creation are possible, these would ultimately end up either making it difficult or time-consuming to join and use the network (which is detrimental to the network by discouraging users), or too easy for malicious users to create more accounts.

One means of mitigation of this is to only permit users to store data, when they themselves are proven to be storing data. This removes the motivation for even the most selfish of users to create multiple accounts, as they would receive no extra storage by having many accounts. Within the context of the MaidSafe decentralised network, this could be achieved in two ways — through proof of storage, and through manager confirmation of this value.

2.6.3 Performance Challenges of Proof of Storage Schemes

Building upon work such as that by Juels and Kaliski [77] and that of Zheng and Xu [78], a proof of data storage could be used to verify that a vault holds the data it claims to, and has not corrupted it. A trivial proof would be for the vault to provide the hash of data. This is unsuitable for actual use, as it is easily replayed — a node holding data could simply remember the hash, and return it each time it was requested, without actually retaining the data. In a DHT, data is indexed by its hash, meaning that the request to verify data must identify the data itself by its hash. As shown by the work of Juels and Zheng though, it is possible to carry out challenge-response style proofs, which would be safe against such attacks.

A limitation of these proofs is that they conventionally require the verifier to themselves retrieve the data, in order to verify the proofs, or to have retained it. While a verifier could pre-compute a number of challenges and responses, and then request them at a later point, this is a hindrance to ad-hoc verification. Being able to verify that a user was positively contributing to the network by storing data in their vaults, without needing to consider the data they held, would be effective and place no additional network overhead on the verification process. Therefore, the high performance proof of arbitrary data storage, as contributed in Section 3.8.2, may be used to allow for the high-performance validation of storage of arbitrary random data. This provides a means for new users to the network to rapidly gain credibility by storing data and proving the storage of this data in an efficient manner, without placing a storage or computation burden on the verifier, an established peer on the network.

2.6.4 Contact and Resource Discovery

An area of intersection between centralised and decentralised services occurs in the field of privacy-preserving centralised services. These services are designed to preserve the privacy of users from the server operator as much as possible. Examples of this include the Signal secure communication application, formerly known as TextSecure.

One common challenge between centralised and decentralised services is that of discovery. Much work has been carried out on the process of service discovery, whereby it is possible to identify what services are available, and access them [79, 80]. Techniques applicable in dencentralised networks have also been proposed [81, 82]. The internet is, fundamentally, a decentralised architecture, and curation and aggregation services such as search engines and hyperlinks have been used to locate services and content for many years [83].

One challenge of discovery is that in order to make an object discoverable, it is typically necessary to broadcast information about its existence and location. For example, to have a website indexed by a search engine, it is necessary for search engines to be aware of the website, either by having a hyperlink inbound to the page, or perhaps by informing search engines about the presence of the website [84]. While in a situation such as discovery of a website, there should be no problem in disclosing the identity of the website, since the owner presumably wishes to make their website available to others, by having it indexed on a public search engine, this is not necessarily the case for all scenarios.

The problem of user or contact discovery is a problem encountered in many scenarios involving applications or services designed to offer social or sharing-related features — it is clearly desirable for a user to be able to locate their existing contacts and acquaintances on the service, without needing to establish contact with each of their contacts, to ascertain if they use the service, and what their username or account identifier is on that service. For this reason, many services, particularly those based around smartphone applications, make use of phone-number based user discovery. At present, this is often carried out by uploading the user's own phone number, and those of their contacts [85]. An intersection can then be carried out on the server, both between the user's phone number and others' contact lists, and between the user's contact list and the phone numbers of other service users. The result of this intersection would then be the list of users to be alerted to having a new contact, and a list of the user's existing contacts using the service, respectively. This naturally raises privacy considerations, as well as posing practical concerns within a decentralised network, where no such trusted intermediary is available.

The current state-of-the-art in privacy preserving contact discovery, even within centralised services, fails to preserve privacy, and the problem of identifying mutual contacts in a privacy-preserving manner remains "an unsolved problem" [86], according to the developer of Signal, a leading encrypted messaging application. State-of-the-art techniques typically use partially-truncated cryptographic hashes [4], relying on the ambiguity of the hash of an identifier to preserve privacy by returning multiple results, which may be selected from by the client. This technique is somewhat similar to that of bloom filters, which do not offer the performance needed for such applications [86], and which also require a central entity to be trusted with the full dataset.

Section 6.2 contributes a solution to this problem, designed for both centralised or decentralised implementations, while preserving privacy in a predictable and modellable fashion.

2.6.5 Ownership-Transferable Data and Records

Conventional approaches to mutable (updatable) data within decentralised networks has typically focused on the storage of data at a known address, derived from a user's public key [63]. This model, such as that proposed within the Bittorrent DHT, places considerable onus on users — it is necessary to either use a different key for each record, or to accept key-reuse, while also incrementing a salt against the key. Neither of these techniques proves particularly desireable — management of unique per-value keys is impractical for most users, and requires users to store keys in a location which may be

[4] https://github.com/SilentCircle/contact-discovery

vulnerable to attack. Alternatively, the re-use of keys means that in the event of compromise, all of a user's data is accessible and able to be modified.

As highlighted by the Bittorrent approach, which is the state-of-the-art in the field at present, as a widely-used DHT implementation, there is no ability to re-key data. In the event that a user believes their keys may have been compromised, there is no procedure through which they may change their keys, since the address of data is derived from the user's public key. Therefore the need to change keys due to potential compromise would lead to the need to re-upload all data to new keys, and remove or clear the previous entries.

Being able to initiate a transition between private keys introduces the opportunity for securely ownership-transferable data, allowing for addresses within a DHT to become tradeable assets, perhaps akin to usernames or DNS-based domain names, by ensuring that a previous owner has no access to the keys necessary to control the data once ownership is transferred. In such scenarios, the ability for the recipient of a key to secure the address using a key held only to them is essential. Section 3.10 therefore contributes an approach to achieve such ownership-transferable data in a decentralised network.

2.7 Decentralised Smart Contracts

Within a decentralised environment, there is inherently a challenge in how to allow mutually untrusting users to carry out business in the decentralised environment. Pothier defined a contract as "an agreement by which two parties mutually promise and engage, or one of them only promises and engages to the other, to give some particular thing, or to do or abstrain from doing some particular act." [87, Section 1]. Fundamentally, Addison identified contracts as being either *bilateral* or *unilateral* [87, Section 1]. In either case, a contract requires two parties to be involved; one of whom makes a promise, and the other to whom the promise is made. Where there are binding obligations on both parties, the contract can be considered bilateral, and where obligations are only binding one party, with the other party not bound by any obligations, the contract can be considered unilateral [87, Section 1].

Within a decentralised storage network, one obvious area where the formation of contracts may be useful is in the provision of storage. By allowing parties to contract others to provide storage to the network, users may enjoy the functionality of providing increased storage to the network, such as having a larger capacity of storage for themselves. While this may be difficult for users of laptops or mobile devices, with limited or no ability to expand their available storage, the ability to contract a third party to provide this would be beneficial to such users. In a decentralised network, there is a challenge posed in how to ensure that a user paying for storage is not penalised as a result of a provider failing to honour an agreement. The contract can be termed *smart*, since it can be validated and enforced automatically by other users within the network. Section 4.2 shows how such contracts can be implemented in a decentralised manner.

2.8 Security and Privacy

Within this thesis, the terms security and privacy are regularly used, when referring to protection of user data, or the usage of such data. Despite a wide range of definitions being found in the literature, the implications of security and privacy shall be considered here, in an inter-related context.

2.8.1 Security

While various definitions of security and privacy exist, security is widely considered to be the overall protection of confidentiality of data, integrity of data, and availability of data [88]. This definition encompasses the needs of a storage system, or indeed a more general networked system, since it provides for being able to reach the data, as well as being assured it has not been tampered with or corrupted, and being able to detect any tampering or corruption. Finally, it covers ensuring that the data cannot be read by an unauthorised party. While there are several other possible properties of a secure system, including trustworthiness, non-repudiation, accountability and auditability [88], these may not always be appropriate — for example, an auditable system must, by definition, reveal some information as to user actions within the audit logs, and the inability to prove what was said may also benefit chat and communications protocols seeking to offer strong security against the compromise of either party in a conversation.

Although security and privacy are often considered separate, it can be argued that one cannot have privacy without security — in light of the many security breaches seen in recent years, resulting in the publication of large quantities of personal information, it is clear that privacy cannot be achieved without security. In the absence of security, unauthorised access to information may be possible, thus violating the privacy of the data.

One solution to the problem of having to trust a company to securely maintain a centralised store of data, including personal data, is for users to not disclose sensitive data directly to the company, and to instead disclose it instead in an encrypted form, therefore only readable by the user, or those they provide the necessary decryption key to. This means that, even in the event of a compromise of a service, or a rogue employee abusing users' trust, the operators of the service have no privileged access to user data.

By only holding encrypted data which is inaccessible to an attacker, the risk of information disclosure in the event of a technical failure is minimised — had Dropbox kept user data encrypted, their failure to verify passwords properly [8] would not have resulted in any meaningful information leakage to those gaining access to accounts during the process. This comes in trade-off with usability however, as users will also lose access to their information if they lose or forget the cryptographic keys or credentials needed to gain access to their information.

While client-side encryption of data stored on remote services therefore offers significant benefits for users, it is not in itself a solution to the problem of security. For client-side encrypted data to be secure, this assumes the client is secure, which is in itself not an easy or safe assumption, given the wide-spread prevalence of highly damaging malicious software infecting users' systems [89]. There are also concerns as to the security of the hardware on systems as well, especially within embedded and mobile platforms, which are now highly popular means of accessing the internet for many users [90]. Finally, the security of the software in use itself, and the distribution channels used to gain access to it, are also of significance, since if the correct software cannot be installed and verified, it is possible for malicious [91] or deliberately weakened software [92] to find its way into use, which could leak keys or otherwise expose users data, despite the service itself being secure and not having access to user data.

There are therefore a wide variety of different security considerations for users storing and retrieving data, both on their own systems, as well as externally.

2.8.2 Privacy

The definition of privacy has been a topic of considerable discussion, with many different definitions proposed and defended. Parker proposed a definition of privacy as "control over when and by whom the various parts of us can be sensed by others" in 1973, before the widespread adoption of ubiquitous computing and the internet [93]. Wilton proposed a more recent definition in 2008, linking identity and privacy, highlighting the importance of controls and consent, and in the separation of unrelated data, such that people may "keep the different spheres of their life separate", and noted that different participants in a discussion of identity and privacy typically had incredibly different ideas as to the meaning of the concepts [94].

DeVries noted that one of the main formal definitions of privacy is from the Fourth Amendment to the US constitution, and the "right of the people to be secure in their persons, houses, papers and effects, against unreasonable searches and seizures" [95]. One of the fundamental challenges raised was that of the analogue nature of this definition — when data about a person is held on a database server, physically outwith the home of an individual, the Fourth Amendment would not be immediately applicable to such a scenario. Indeed, precedent has since been established that the Fourth Amendment protections do not apply to data held on a third party system, on the argument of the so-called third-party doctrine, in that “a person has no legitimate expectation of privacy in information he voluntarily turns over to third parties.” (442 U.S. 735 (1979) Smith vs. Maryland).

Part of the challenge of applying an analogue privacy law is due to the difficulty in applying the concepts of *search* and *seizure* to digital data — seizure has been ruled by the Supreme Court of the United Stated to be “some meaningful interference with an individual’s possessory interests in that property” [96]. With an analogue possession, its physical seizure may result in deprivation of access by the owner, in the same way that theft deprives the rightful owner of their property. In contrast, when electronic computer data is considered, the potential to create a bitwise digital clone of the data raises new questions and challenges. Specifically, when a search or seizure would be required for evidence gathering, there is no requirement to deprive someone of possession of their data, since a digital copy may be made. This may also be carried out on data being transferred between computers, without physically gaining access to an individual’s property and carrying out physical searches for data, as discussed by Kerr [97].

Privacy in the digital age, from the perspective of US law, was described by DeVries as being split into two freedoms — the freedom to be left alone, and the concept of informational privacy to cover data and its use. While the former was argued to be enshrined by the protections against government intrusion into life in the United States Constitution, it was to cover “protection of autonomy and free choice”, whereby decisions are based upon whether the claimant had a “reasonable expectation of privacy” and if appropriate justification was given by the party or government entity breaching it [95].

Kerr has highlighted, however, the challenge over what counts as a search or seizure — any violation of a “private space” counter to a “reasonable expectation of privacy” is considered a search under the Fourth Amendment, and the physical act of taking evidence from the scene for use at trial is a seizure [97]. Within the context of digital data, this would indicate that the taking of a computer system or hard disk drive would be considered a seizure, as possession of the device would be lost. However, the seizure of data digitally, without physical interference, as a result of downloading of data from a remote server, was found to not amount to a seizure, as it was not altered or made unavailable [97].

Specifically in the context of digital data, Kerr highlighted an important consideration as to seizure, given computers work by making copies of data — when an email is sent across the internet, it is split into packets of data, and these packets are transmitted toward their destination. At various intermediate points, they are processed by routers, and are re-transmitted. Many actions also inherently create copies of data — for example, an email server will create a copy of an incoming email to place it within a user's inbox. Kerr's thesis of when data copying presents data seizure has attempted to reconcile this by arguing that a seizure occurs when the "intended course of possession or transmission" is interfered with, and it was not already known to the person accused of the seizure. The concept of privacy within the cloud has also been considered by Robison in an analysis of the online privacy in the context of US legislation, specifically where cloud services are used as communications services, as well as for remote computing services [98].

2.8.3 Privacy in Legislation

Within the United States, where many online services are based, there is no single law enshrining the right to privacy in all contexts — the Fourth Amendment, as discussed previously, offers protections against government search and seizure, but makes no consideration as to private or corporate search or seizure. The 1974 Privacy Act defined the FTC's "fair information practices" (FIPs), although this covers government compliance, and only to information issued or collected by government, and has many limitations [95]. Other laws covering privacy include the Fair Credit Reporting Act, the Computer Fraud and Abuse Act, the Electronic Communications Privacy Act and the Children's Online Privacy Protection Act, which each cover only a small and limited area of privacy. A more complete list is given in *An overview of privacy law*, by Solove & Schwartz [99].

From a European perspective, however, there are strong privacy protections in place as a result of the Data Privacy Directive (95/46/EC), a European directive to member states to pass a local law covering the points of the directive, themselves based upon the OECD's 1980 principles for the protection of personal data [100]. As a result, governments within Europe have had to be much more proactive in introducing legislation and regulation around user privacy [101]. Indeed, as Fromholz discussed, the USA and Europe have vastly different approaches to the regulation and control of individual privacy — European regulation has been proactive, based on pre-empted needs for regulation, and have led to broad privacy protections being in place. In contrast, US privacy legislation is typically narrow, designed to legislate only as a reaction to a particular incident — as stated by Schwartz & Reidenberg, medical and direct marketing data stand out as particularly lightly regulated areas [102]. Swire & Litan have also reviewed the Data Privacy Directive, highlighting the significant differences in culture and approach to privacy, highlighting how people in the US are typically more concerned about the abuse of personal data by government, from centralised databases, and that this has fed into the majority of US privacy legislation [103]. In stark contrast to this, the European Directive is focused more on preserving user privacy against companies holding or gathering personal data, and on ensuring that appropriate consent is given by users [103].

The point of intersection between the two seemingly antithetical and perhaps incompatible approaches to privacy was what was referred to as the Safe Harbor agreement, brokered between the United States and the European Union in 2000, shortly after the introduction of the EU Directive [99]. This agreement was made, as the US did not have privacy laws of a sufficient level to meet the requirements for "adequate" privacy protection, but companies based in the US wanted to have access to the European market, and customers residing there, without having to tighten their own domestic legislation to the extent required by the EU [104]. Indeed, the EU-US Safe Harbor agreement has been described as a compromise between the typically self-regulated approach to privacy by US companies [105], and the much stricter European legislation [104].

The Safe Harbor agreement aimed to remove the challenges faced with the EU's comparatively heavily regulated handling of private data (which is established as a "fundamental right" [104]), in comparison with the relatively hands-off approach taken in the US, where privacy protections are typically implemented voluntarily by companies, in order to prevent potential lawsuits [104]. Despite this, a European court ruling recently led to the effective suspension of the Safe Harbor provision [106]. In this case, Schrems brought a case against Facebook Ireland Ltd. on account of its transfer of EU citizens' personal data to the United States, and its storage there. The case centred around Schrems' complaint that Facebook's transfer of his personal data to the United States did not offer sufficient protections, and that he wished to exercise his right to prohibit the transfer of such data, on account of surveillance activities being carried out by public authorities within the United States.

The Irish High Court found that, since US law did not give EU citizens a right to be heard, and that oversight of intelligence services was carried out through secret procedures, it did not offer adequate protection. The end result was that Decision 2000/520, which stated the European Commission "may find that a third country ensures an adequate level of protection", and established that the Safe Harbor program offered such protections, was reversed. This meant that data transfers under Safe Harbor are no longer automatically valid, and that legal challenge may be made to data being transferred to the United States by European companies [106].

This case has highlighted the importance of legal protections within the operation of services, and the difference in approach between the US and EU's respective approaches to privacy. The use of encrypted, decentralised services presents a solution to this, by allowing a developer in the United States to provide a service to users worldwide, without having to navigate regulatory challenges in order to store user data. By offloading the storage of encrypted (and thus not personally identifying or sensitive) data to a decentralised network, the service will be more reliable and resilient, as well as more private, while ensuring the service preserves user privacy, irrespective of location or jurisdiction.

2.8.4 Privacy in Relation to Services and Storage

As highlighted above, data privacy is considered as a matter for self-regulation within the US, with the FTC encouraging companies to self-declare their own policies and procedures, allowing customers to choose from services based on these declarations. Therefore, such self-set policies and agreements become an area of interest for those using or accessing services.

Privacy policies, terms and conditions, and other legal documents form a near-universal part of the experience of using connectivity-based services today. Virtually every website, mobile app, and even physical service provider has an agreement of this form, to which users are required to agree, in order to make use of the service. Often, however, these agreements are stated to be implicitly accepted by accessing or using a service, which gives rise to a number of considerations surrounding the validity of these agreements. Online agreements typically take the form of either a click-wrap [107], or browse-wrap [108]. These names are derived from an early form of software-related agreement, referred to shrink-wrap, whereby a user was held to have accepted a software End User License Agreement (EULA) by opening the shrink-wrap seal on the physical packaging itself [109].

The original premise of these policies was that, with the rise of general-purpose computer software being sold, it would be impractical for every user of a piece of software to individually negotiate a contract with the company providing the software. The ability to offer a standardised agreement, which users would automatically indicate acceptance to through the use of the product or the unsealing of the packaging, created a concept whereby a standard-form contract could be offered and established between users and the providers of software [109]. As web-based services have gained popularity, a number of new considerations arise, specifically around how these policies are put in place, and how variations or changes to these policies are handled, since these policies are effectively legal contracts. "signed" through clicking a box, or indicating acceptance.

These considerations have been highlighted by recent trends in how online businesses are run. For example, in recent years there has been a tendency for companies to build their businesses around the prospect of making money as a result of data gathered from the users of an (otherwise) free-to-use service. Indeed, as Bruce Schneier stated in a conference talk in 2010,"Don't make the mistake of thinking you're Facebook's customer, you're not — you're the product," [110]. With the rise in free (at point of use) services on the internet, designed to encourage users to engage with them for the purpose of gaining a larger user-base, which itself is then claimed as an asset by the company for the purpose of its valuation [111]. This highlights an interesting consideration for privacy — it is in the interests of a company to ensure its own value is as high as possible, and if a company's value is based upon the information it holds about its users, there will be a fundamental risk to user privacy in these scenarios. This has been showed true in a number of scenarios where companies which held user data were liquidated, and attempts were made to sell or otherwise liquidate user data as an asset. For example, during the bankruptcy of RadioShack in the United States, an attempt was made to sell customer data including names, email addresses, addresses, and phone numbers [112], as well as information about purchase values, and stores visited [113]. This fell in direct contrast to the company's own privacy pledge, which stated "we will not sell or rent your personally identifiable information to anyone at any time", and that "we pride ourselves on not selling our private mailing list" [114].

As a result of representation made by other companies, as well as various US states, the process was scaled back, to no longer include phone numbers, and only include email addresses of customers who had made a purchase in the previous 2 years [115]. Nonetheless, the approved sale was in direct contradiction to RadioShack's own privacy policy, showing that such policies are not necessarily legally enforceable by customers, in the event of a company's bankruptcy — the data was offered for sale in a public auction [112].

Online services are recognised as being largely self-regulated, with regard to the handling of personal data (through privacy policies), which is why the FTC generally seeks to hold companies to account by ensuring they honour their own (self-set) privacy policies [105]. Significant in this case, was that sale was inhibited as a result of the company's own privacy policy, and not on account of privacy legislation [115]. This was also seen in the case of Toysmart, which was in a similar situation— Toysmart reached a settlement with the FTC, agreeing they sold customer information after stating it would never be shared with third parties. Jodie Bernstein, the director of the FTC's Bureau of Consumer Protection, stated "Customer data collected under a privacy agreement should not be auctioned off to the highest bidder" [116]. Significantly, the FTC ruling also stated "The settlement also protects customers of Toysmart from unilateral privacy policy changes in the future by a bankruptcy purchaser. Any change in the original Toysmart policy will have to be approved by consumers on an 'opt-in' basis before the successor company can make such a change". This highlights an important area for consideration from the perspective of cloud and web-based services, namely that of privacy protections for users of services in today's increasingly centralised network services.

2.9 Human Factors

One of the fundamental challenges of building services, both centralised and decentralised, is in securing user credentials and keys against theft of unintentional disclosure. These arise as a result of the human factors within the implementation of secure systems — a system may have a proper cryptographic implementation, yet offer no actual confidentiality if a user reveals their passwords through social engineering, or re-uses them across other insecure services.

2.9.1 Passwords as Credentials

When the security of a system is placed in the hands of users whose role is not specifically related to security, the risk of serious vulnerabilities in practical security arise, such as seen in the case of Phone House, an independent Dutch mobile phone retail outlet, co-located within Media Markt, a large electronics store. In this case, multiple serious failures in operational security, including amongst others the writing down of passwords on post-it notes and the use of shared unencrypted documents containing (weak) login passwords to external portals, allowed a customer to gain access to the personal data of over 12 million Dutch phone users [117].

Fundamentally, this attack was successful as a result of poor operational security awareness by employees, and the reliance upon password-based logins for services containing highly confidential information. Passwords are, by their nature, inherently vulnerable to replay attacks, shoulder-surfing attacks, and social engineering attacks. In a typical password-protected system, having knowledge of the password is sufficient to allow ongoing access to the protected resource, until the password is changed, either as a result of routine, or as a result of the account owner becoming aware of the unauthorised use of their account.

Many alternatives to password-based login have been proposed, and indeed exist in currently available implementations. Client certificate authentication was defined in TLS 1.0 [118], providing a means for websites to authenticate a user via a public-key certificate held within their browser. Two-factor authentication protocols are also available to augment password-based login systems. One commonly used approach is that of TOTP [119], which generates a short, easily typed, time-based numeric one-time password, based on a symmetrically shared secret between the client and the server.

With the rise in use of touchscreen-based devices, the difficulties of password entry are also becoming increasingly clear. Conventional advice is to increase password length, and increase password complexity through the addition of numbers and symbols, although these measures face usability challenges on devices not designed for complex text entry, such as smartphones [120]. Fundamentally, password authentication is a process of proving knowledge of a secret, and the server-side validation of this secret. While zero-knowledge password protocols have been proposed and implemented [121, 122, 123], significant web services and applications continue to favour transmission of user passwords in plaintext to the server for validation, as demonstrated in Section 5.2.3 with regard to research carried out on Google's platform.

Within the context of a decentralised network, such as MaidSafe, the security model is based around the assumption that a user can derive an identity key as a result of their own credentials. Improving the handling and secure storage of such credentials will therefore significantly improve the security of a storage solution, as a result of reducing the exposure of keys to human-induced weaknesses.

2.9.2 Key Storage

When a given message is encrypted by a key, it is naturally necessary for the key to be retained, to recover the original message. If the scenario of file storage is considered, the ciphertext may be assumed to be stored elsewhere, and therefore only the key need be retained. Nonetheless, it is essential for recoverability of data that the key is stored in a manner which retains its confidentiality. Otherwise, an adversary gaining access to the key may retrieve the data and decrypt it. One approach to this problem is be to create an encrypted store, within which such keys may be stored, which is the approach taken by password managers. In this case, however, the process of encrypting this store would generate a new key, which must be retained to gain access to the store. This new key then needs to be held confidentially, but with the knowledge that loss of this key will remove access to the entire contents of the key store. The perceived risk to the user of the loss of this key is therefore potentially higher — loss of all their data, or a significant proportion of it, would clearly be a worse scenario to recover from than the loss of a single file.

Therefore, both a collection of many keys, or a master key used to gain access to many keys are of high value to a user, and users will take measures to prevent their loss. With the limited ability and perhaps willingness to memorise large quantities of passwords [124], it is impractical to expect users to commit to memory large numbers of unique keys. Indeed, previous large-scale studies have found that typical user passwords may have between 37 and 51 bits of entropy [125]. Such passwords are not sufficient to use directly as cryptographic keys, and indeed should not be considered as keys, due to their limited randomness, and limited character-set from which the characters of a password are chosen [125].

A visible example of this effect is seen in the Bitcoin cryptographic currency. Bitcoin offers users a truly decentralised approach to holding money in a wallet under their control, which can only be spent through knowledge of a private key [126]. Despite this, large numbers of users choose to hold their coins in centralised exchanges or online wallets, offering backup and storage of their account private keys [127]. This continues, even after many security breaches, leading to personal information compromise or theft of coins [128, 129]. This may be as a result of the perceived "safety" offered through the use of such a service, since loss of a wallet key or password will not result in the irretrievable loss of their money.

Applying the same logic to storage services highlights that there may be a clear trade-off between security and usability which should be considered. If users feel they may easily lose their keys, it is likely, given the precedent from Bitcoin, that users may store their keys in potentially insecure manners, such as escrowing them with a third party perceived to be trustworthy, in order to mitigate against loss of their keys, and thus data.

2.9.3 The Value of Successful Attacks

Recent years have seen considerably numbers of high-profile security breaches of personal and confidential data, which was entrusted by users to companies. In one case, 80 million customers of a US-based health insurance provider had personal details including names, addresses, dates of birth, social security numbers, and employer details and income accessed [6]. In another, the fingerprint records of 5.6 million US federal employees with security clearances, and 21.5 million personnel records for other federal employees, including highly personal security clearance questionnaires, were accessed [7]. These attacks have since been attributed to a foreign attacker, although the precise motivations remain unclear [130].

It is clear, however, that one key ongoing challenge faced by companies is that of how to securely store information, preventing it from being accessed by unauthorised parties, while ensuring it remains usable when authorised access is required. Usability here is a considerable challenge, since invasive security measures may require significant changes or actions on the part of the user. Many security systems in use today require the user to act as the decision-maker when there is doubt or uncertainty as to the security of an operation. For example, the widely used Secure Socket Layer (SSL) scheme is implemented in web browser to provide secure connections between user web browsers and servers. In the event of an error being encountered, the default behaviour of SSL is to fall back to asking the user to make a decision, and presenting the details of the error which was detected. Previous work has investigated the efficacy of such a process, and highlighted the extent to which users may make incorrect decisions in situations where certificate validation fails, recommending instead that users not be relied upon to make decisions where an unsafe situation has been detected [131]. The reliance upon a human user as a decision-maker for trust has also presented itself as a problem in scenarios where a user is not present, such as in embedded systems or non-interactive scenarios where there is no web browser for a user to interact with [132].

2.10 Security of Implementations of Current Services

The Android operating system is an always-online, cloud-enabled operating system [133], released as an open-source core, referred to as the Android Open Source Project, in addition to a proprietary and commercially-licensed suite of Google-branded software, commonly referred to as Google Mobile Services (GMS) [134]. GMS is a centralised service, operated by Google, in-keeping with the description given in Section 2.1.1.

Google reported in its May 2014 I/O event that there were more than one billion active users of Android. Given the near-ubiquity of Android devices, it is worth bearing in mind that weaknesses present in the platform, particularly in the closed mobile services components, which are not modifiable by manufacturers in the same way as the open components of the platform, may expose over a billion users to attack [135].

Previous works have explored the check-in process carried out by mobile devices using older versions of Google Mobile Services [136], and focused on a number of privacy considerations with regard to data transmitted to servers operated by Google.

To preserve the security of information transferred between a client device and the corresponding back-end service, Transport Layer Security (TLS) is used. TLS is the current state-of-the-art transport protection protocol, incorporating authentication of identity of the server, in addition to encryption of the data transferred across the link, and is widely used in online banking and other websites and applications. It is commonly referred to as Hypertext Transfer Protocol (Secure) (HTTPS), and recognised by end users through the presence of a padlock icon within their web browser.

2.10.1 Overview of Transport Layer Security (TLS)

Communications between Android devices and the Google servers were protected using TLS. TLS is designed to authenticate the remote server, ensuring it is operated by the party claimed in the domain name, and to then provide protection against both active and passive attackers sniffing or intercepting traffic on the wire, through the use of encryption [118].

To monitor and carry out research on traffic protected using TLS, an interception system can be configured. For Android devices, a dedicated Wi-Fi network is created, and the Android device's network settings are manually configured, such that a Linux computer running `mitmproxy` software is set as the default network gateway [137]. This ensures that all network traffic from the device is routed via `mitmproxy`, which then forwards it through another network interface to the internet, after carrying out false TLS handshakes. A new certificate authority (CA) must be generated, and the public key of this must be manually added to the system partition of an Android device. This allowed the proxy to generate certificates believed valid by the operating system. This simulates the same scenario as a compromised publicly trusted Certificate Authority (CA).

2.10.2 Previous CA Compromises

The CA system is far from perfect, since the compromise of a trusted CA will permit rogue certificates to be issued. Despite procedural and policy-based measures in place to attempt to prevent this, there have been a number of high-profile compromises of CAs in the past, indicating that TLS is far from perfect when considering dedicated attacks by parties able to compromise, or already in control of, a trusted CA.

The default Android 5.1 operating system image, shipped by Google, was found to ship with 162 trusted root certificate authorities. Such certificate authorities may issue certificates for any domain, and recent efforts to monitor issuance and use of certificates have highlighted a number of scenarios where certificates have been mis-issued. For example, Symantec recently issued an extended-validation certificate for Google.com as part of a testing process, breaking policies CAs are required to abide by [138]. On another occasion, a trusted intermediate certificate was issued by CCNIC, and used by MCS Holdings in a man-in-the-middle proxy device [139]. In yet another occurrence, the Indian National Informatics Centre issued certificates for Google domains [140]. Similarly, the French ANSSI certificate authority was found to have falsely issued certificates for Google domains, which were then used within commercial devices to carry out monitoring of network traffic [141].

It is therefore clear that certificate authorities are a viable target through which false certificates may be generated, given these situations, as well as the external attacks on Diginotar and Comodo, leading to the issuance of certificates for sites such as Google, Tor and Yahoo, which may have been used to intercept traffic of Iranian internet users [142]. With the compelled certificate issuance attack [143], the concept of coercion or compulsion being used against a valid CA to issue a certificate for a third party website was also introduced, offering another threat which client endpoint devices may be subjected to.

2.10.3 Certificate Pinning

In light of the issues with the CA system identified above, certificate pinning is a technique used to attempt to reduce the exposure of applications reliant upon CAs for server authentication, by constraining the CAs permitted to attest to the identity of the service in question. Certificate pinning therefore acts as a means for a client device or application to restrict the CAs which will be trusted to issue certificates for a given domain [144]. Under certificate pinning, an application or web domain may be configured to only accept certificates issued by a given CA, or to only accept a given public key for connecting to that domain. The release notes for Android version 4.4 stated that it introduced an implementation of this to protect users against connections to false Google certificates [145].

In a working implementation of certificate pinning, a whitelist of permitted certificates (incorporating public keys) should be loaded to clients prior to being shipped — to prevent a compelled certificate issuance attack [143] from being carried out against users, it is important that a list of public keys is stored, rather than a list of permitted certificate authorities, otherwise the same certificate authority could be compelled to issue an adversary a certificate for the site. Support for certificate pinning was introduced in Android version 4.2 [146].

In Section 5.2, the security of the deployed Android certificate pinning implementation will be investigated as a contribution to this thesis, to investigate the security offered by a real-world deployment of this technique. This is significant to decentralised network security, since in a decentralised network, there is typically no third party on the network which can be trusted. In contrast, certificate pinning is implemented as a means to restrict the set of third party CAs which are able to issue trusted attestations about the authenticity of a remote centralised server. With certificate pinning being an action hidden "behind the scenes" within applications, it is therefore essential from a user perspective that it operates as expected and described, since it is difficult for a user to verify this.

2.11 Conclusions

This chapter has detailed the background and context within which this work exists. Related work on decentralised networks has been explored, and some of the challenges posed in the design and implementation of decentralised networks have also been highlighted. An overview of the MaidSafe network, the subject of chapters 3 and 4, has been given, including the necessary background material on the operation of the network, its addressing structure, and the main principles of operation of the network.

In addition, some background on decentralised and smart contracts has been given, and on the concepts and definitions of security and privacy, both from a technical and legal perspective, since the main focus of this work is on the preservation of security and privacy of personal data. Finally, the challenges of provision of practical security have been explored, specifically around human factors such as the use of passwords for security, and the challenges of cryptographic key storage and use. This will be used as background to the problem of secure identity management, when considered in Chapter 7. Finally, some background of TLS has been considered, within the context of communication with mobile application Application Programming Interface (API)s. The issues of trusting a wide range of CAs has been introduced, and the operation of certificate pinning as a means to restrict the CAs trusted by a given application or website has been considered.

Chapter 3 will now present the first contributions of this thesis, investigating the security of the MaidSafe network, and how such a network can be protected against attackers flooding the network with new members, without compromising the decentralised nature of the service.

Chapter 3

Security of the MaidSafe Decentralised Network

3.1 Introduction

The MaidSafe network is a decentralised, distributed network, intended for the storage and retrieval of files and other user data. Background to the MaidSafe network is given in Section 2.5. Any user may join the MaidSafe network and provide storage resources through a ‘vault’, and make use of the network to retrieve or store data, through a ‘client’. This chapter shall provide a thorough overview of the MaidSafe network, its principles of operation, as well as the security model upon which it was designed. This chapter shall explore the operation of such a network, which in turn explains many of the requirements of a decentralised network. It shall then introduce a number of contributions with regard to the security of the network, and present a number of new potential attacks against a decentralised network.

3.1.1 Chapter Contributions

This chapter presents the following 5 contributions.

- A security evaluation of the MaidSafe network, highlighting new weaknesses and design considerations within decentralised network architectures.
- An improved cryptographic implementation of the MaidSafe self-encryption algorithm, offering significantly improved performance, while using the same cryptographic constructs.
- The identification and demonstration of practical attacks against the self-managed nature of the MaidSafe network, including a distributed architecture able to carry out rapid attacks against the network through pre-computation.

- A novel high-performance proof-of-storage algorithm, allowing for the low-overhead verification of contribution of storage to the MaidSafe network, without the identified vulnerabilities of the current implementation within the MaidSafe network.

- A decentralised protocol for ownership-transferable data, allowing for mutable data to be stored under selected addresses on the network, while also allowing for the transfer of control of the address to a new owner or key, permitting transfer of ownership and re-keying of a user's keys.

3.1.2 Terminology Notice

Throughout the remainder of this chapter, the terms vaults and clients will be used to refer to the storage and data retrieval entities respectively. The term 'node' will be used as the collective term for both clients and vaults, where no distinction is necessary, such as when considering the design and architecture of the underlying distributed hash table. Vaults will typically be considered as fixed computers which remain connected to the network the majority of the time. Clients are devices which may be mobile or fixed, and may not always be online. In many cases, a single computer will run both a client and vault, but it is not necessarily assumed that both are running on the same system.

3.2 MaidSafe Self-Encryption Algorithm

The self-encryption implementation presented by MaidSafe is a somewhat modified version of convergent encryption, through the use of block-based chaining. This section therefore analyses the implementation of convergent encryption within the MaidSafe network, as introduced in Section 2.5.1.

One initial weakness of the MaidSafe system was identified and reported through concept review — the system did not feature ciphertext authentication — cipher modes proposed [61] did not feature inbuilt authentication, or use AEAD (Authenticated Encryption with Associated Data) techniques, such as encrypt-then-MAC.

The original MaidSafe proposal for self-encryption was described as part of a whitepaper [61]. The current implementation is based on this original proposal, although it now uses the Salsa20Poly1305 authenticated cipher, rather than the AES-256 cipher as originally proposed. Therefore, the lack of ciphertext authentication within the original AES-CBC implementation has been addressed as a result of reporting this weakness.

3.2.1 Analysis of the Encryption Implementation

This section provides an analysis of the current implementation of the MaidSafe self-encryption algorithm [1], as of June 2016.

Firstly, plaintext data of non-trivial size is split into n chunks (C_0 to C_{n-1}). If a file is split into chunks, it is split into a minimum of $n = 3$ chunks. Chunks are a maximum of 1 MB in size, and a minimum of 1 KB. In the event of a file being smaller than 3 KB, it would be directly encrypted into the data map structure, rather than split into chunks.

The key material for chunk C_x is selected as the first 32 bytes of $H(C_{x-1})$, and the nonce is selected as the subsequent 24 bytes. Note that for the first chunk of the file, where $x = 0$, C_{x-1} is selected as the last chunk of the file.

The padding chunk for chunk C_x is the full hash of the chunk C_x (64 bytes), followed by the final 8 unused bytes of the hash of chunk C_{x-1}, and the full hash of chunk C_{x-2}.

After instantiating the Salsa20Poly1305 cipher using the key and nonce, both of which are deterministic and based upon the file's contents, the resulting ciphertext is XOR'd with the padding chunk — the padding chunk is simply repeated until it reaches the length of the ciphertext output. This output becomes the data to be stored on the network, and its cryptographic hash is taken as the address under which the data is stored, since data is stored at the address of its hash. It is worth noting that this XOR process does not add to the security properties, and is thus extraneous.

Finally, the keying data needed to recover the file is stored in the form of a data map, which contains the plaintext hashes of each chunk, and the resulting ciphertexts of each chunk. The hashes of each plaintext chunk act as the necessary keys to remove the padding process carried out, and carry out decryption, while the ciphertexts are the addresses from which the encrypted data must be retrieved form the network.

Figure 3.1 represents this process, showing the selection of key, nonce and obfuscation chunk from the hash outputs of two adjacent chunks.

This shows that block-level deduplication is taking place, and that deduplication can be achieved through a single file. Significant, however, is that there is potential information leakage in two ways, as a result of the MaidSafe encryption implementation. The first is the well-documented principle of known-plaintext attacks, as discussed in Section 2.5.1 whereby any user of the decentralised storage network may attempt to identify if existing known plaintext data is present on the network [147]. Since any party knowing the plaintext of a file may generate the same ciphertext (per the definition of convergent encryption, where the same key and nonce are used for each chunk, given the same original plaintext), it is possible for a party knowing a plaintext to locate it on the network as a ciphertext, and thus potentially monitor it, or users accessing it.

[1] https://github.com/maidsafe/self_encryption

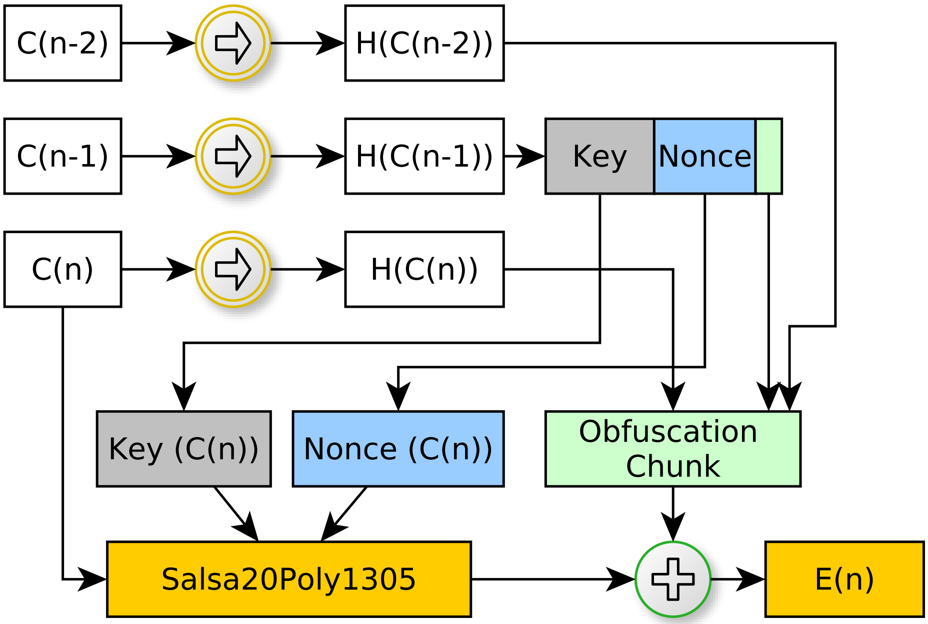


Figure 3.1
Representation of MaidSafe self-encryption process for chunk C_n

The second potential attack was identified as a result of analysis of the implementation and its output, given the nature of the MaidSafe implementation of block-level deduplication. Since a chunk has influence over the ciphertext of two other adjacent chunks, it appeared that it should be possible to identify files sharing a sequence of three common blocks. By way of example, three files were created, each of exactly 6 MB in size, designed to result in 6 chunks of ciphertext. The first file contained all zero bytes, the second was identical with the first byte set to `0xFF`, and the third was identical to the first, with the last byte of the file set to `0xFF`. The purpose of this test was to demonstrate that inferences could be drawn between files that were similar to each other. Here, the the plaintexts (with the exception of the final byte) can all be considered to be prefixes of each other. One of the principles of a strong cipher implementation is that identical blocks of data should not encrypt to the same ciphertext output [148]. Specifically, the hypothesis posed here is that patterns between plaintext chunks may be passed through to the resulting ciphertext blocks.

The first file (all zero bytes) was encrypted into a single block, since the hash of each block was identical, resulting in each identical block being encrypted in an identical way. For the purpose of clarity, cryptographic hash outputs are given in a truncated form; all were distinct, and no collisions occur in this truncated form for presentation. The first file encrypted into the ciphertext block `5dc38`. This was therefore the block output for a 1 MB block of bytes with value `0x00`, with both adjacent blocks also being `0x00`.

The second file, identical with the exception of the first byte being `0xFF`, encrypted as follows. The first ciphertext block was `f085f`, the second was `5b725`, the third was `6924a`, and the remaining blocks were again `5dc38`. This was as expected, since for a varied chunk C_1, chunks C_1, C_2, C_3 would be expected to differ from the remainder, which would be otherwise unchanged from the previous test.

Using the third test file, where the last byte was set to `0xFF`, it was predicted that the last, first and second chunks would be altered compared to the original test, and this was confirmed, with the ciphertext output hashes being `d1bf4`, `aad69`, three blocks of `5dc38`, and a final block of `8cd03`. Figure 3.2 illustrates these three experiments, and highlights the propogation of variations in plaintext blocks through to ciphertext blocks. Specifically, the presence of unmodified blocks shows that the self-encryption process specifically does not cascade variations through the resulting ciphertext, allowing for analysis and comparison of ciphertexts.

Therefore, the presence (and equality of) three adjacent identical blocks could be identified within the file in question, without knowledge of the original plaintext data. This confirmed that the output of the convergent encryption algorithm, based upon the implementation by MaidSafe, would make it possible to reveal the presence of three known adjacent plaintext blocks, as well as identify patterns at block-level within plaintext files. For a file with chunks towards the 1 KB lower chunk limit, it is highly likely that, in some contexts, the structure of the file may be repeatable between files. This allows for the revealing of information both about the internal structure of similar files, as well as between different files.

This information leakage, however, is unavoidable if one wishes to achieve global, rather than per-user, deduplication, as used by MaidSafe. Convergent encryption necessitates that a given plaintext encrypts to the same ciphertext, and block-level deduplication requires that a small change in a file does not result in cascading changes throughout the file, which would render deduplication ineffective.

3.2.1.1 Critical Evaluation of Approach

Since the MaidSafe proposal contains no facility for re-keying of data, with the key to a given block of data derived only from the block itself, and the next two blocks, this highlights that known-plaintext attacks may be leveraged across third party users' data, but at block level, rather than at file level. The advantage of this process, however, is that deduplication may operate at block level, rather than at file level, although this would not likely result in as much inefficiency as one may expect. Meyer and Bolosky carried out a study of over 800 desktop computers having their filesystems backed up, and found that whole-file deduplication would achieve about 75% of the savings of even

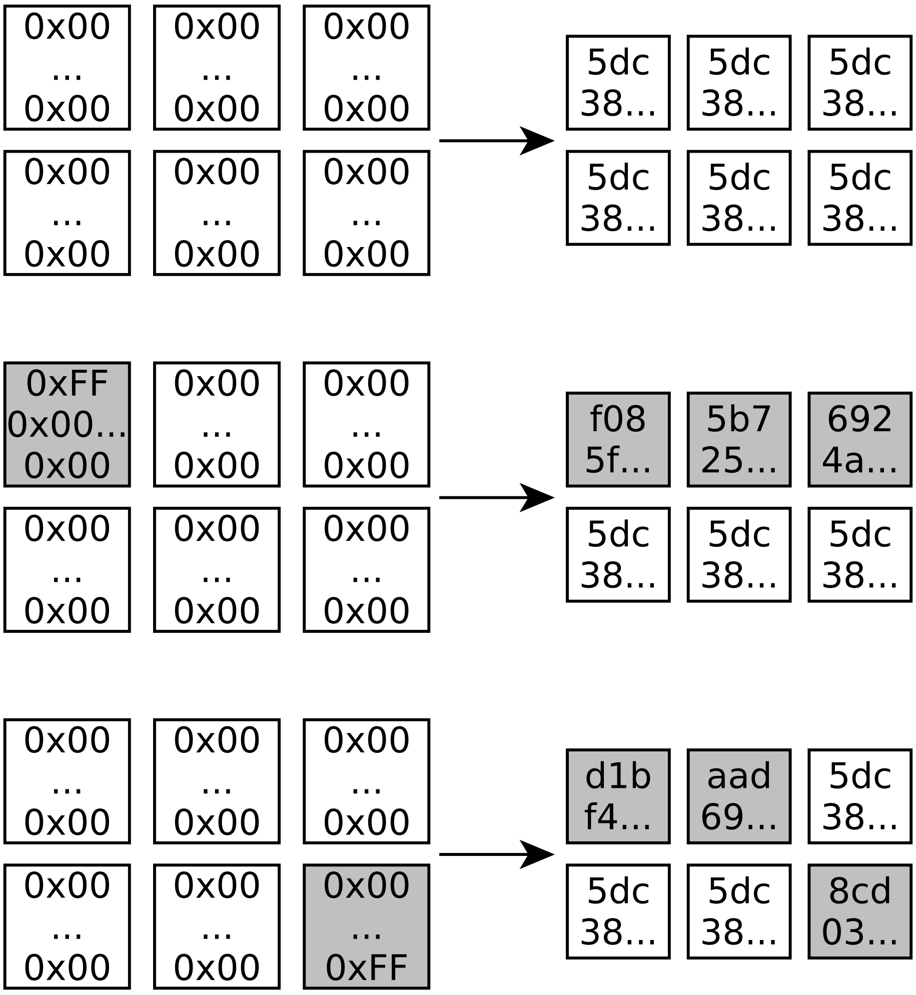


Figure 3.2
Correlation between input blocks and output ciphertexts undergoing self-encryption

the most aggressive (and thus efficient) block deduplication, and that for four weeks of full image backups, whole-file deduplication achieved 87% of the savings made using block-level deduplication [149]. Also worth noting is that Meyer and Bolosky's study used much smaller blocks of data, with a maximum block size of 128 KB considered.

In contrast, the MaidSafe approach considers blocks to be typically 1 MB each, for files larger than 3 MB, with three common blocks needing chained together to result in the same ciphertext being produced, and therefore an invocation of deduplication. Intuitively, using a significantly larger block size will result in fewer savings — to match a block between files, each must contain an entire block, relative to the chunk boundaries, which is identical. In MaidSafe's implementation, this requires three adjacent identical blocks of perhaps 1 MB each, before deduplication would offer savings. The use of variable-length deduplication could also improve the efficiency of deduplication with such large blocks [2].

Therefore, the use of file-level deduplication, may expose some information as to the structure of users' files, and their commonality with other files. From previous work, it has been shown that file-level deduplication offers relatively high efficiency compared to block-level deduplication, with significantly smaller blocks, indicating that the losses of using file-level deduplication may be minimal.

By accepting that deduplication would only be carried out at file-level, it is also be possible to remove the potential leakage of information as to the relation between ciphertexts, without any disadvantages compared to the current approach. Specifically, it is clear from the process used by MaidSafe, and evaluated above, that it is possible to begin decryption from any location of the ciphertext, provided the appropriate keys are known. It would also be possible to significantly simplify the above process, removing the stage of "obfuscation" from the MaidSafe proposal, through the use of a stream cipher. By basing the stream cipher key and nonce/initialisation vector from the hash of the entire file, a secure stream cipher output can be derived using knowledge of the entire file's contents, rather than an individual block's contents. Indeed, the Salsa20Poly1305 cipher already used by MaidSafe's current implementation is a suitable authenticated stream cipher which could be applied across the file. The ciphertext could then by chunked as required, and the resulting chunks hashed to obtain their addresses on the network. This would remove the need to carry out a per-byte XOR process against a varying output, which has been found to be a significant performance overhead [150], and would result in a ciphertext which could only be identified by a third party possessing the full contents of the plaintext, rather than merely a subset thereof. The Salsa20 stream cipher has a constant-time seekable output, allowing for a

[2] https://restic.github.io/blog/2015-09-12/restic-foundation1-cdc/

given chunk of data to be decrypted in constant-time, without requiring access to previous chunks, provided knowledge of the key and the offset of the chunk in relation to the start of the stream cipher [151].

3.2.2 Improved Performance of Convergent Encryption Implementation

In an attempt at improving the performance of the MaidSafe Rust implementation of self-encryption, which was found to perform poorly, even when compiled as a release binary, the algorithm was implemented in C. Libsodium was used as the provider of all cryptographic functions, in order to ensure a well-reviewed implementation was used. The only algorithmic modification taken was to remove the padding process — the obfuscation of data chunks by XOR'ing against other keying material from related blocks raises a potential weakness if an adversary were able to obtain intermediate ciphertexts through a side channel, as it would potentially expose keying material for other blocks. By removing this step, which is not necessary for the security of the Salsa20Poly1305 cipher construct, the risks of XOR'ing with raw keying material can be eliminated.

To ensure an accurate comparison between the MaidSafe implementation of convergent file encryption, and this improved version, this implementation was configured to operate in the same way. Firstly, the same chunk size selection algorithm was implemented, to avoid any disparity due to filesystem overheads if different chunk sizes were used. Secondly, both systems were configured to output to a `/tmp` ramdisk on the same Linux system, ensuring that storage performance was not a constraint on overall performance. The MaidSafe implementation also carried out an internal self-check on the resulting ciphertext, and a similar check on the ciphertext output was implemented in this alternate approach, to ensure the task being carried out was comparable between both instances.

Since the MaidSafe implementation was single-threaded, the improved implementation was also implemented only as a single threaded application. For both applications, the exact same test data was used; a single 100 MB file containing pseudo-random data, retrieved from `/dev/urandom` using the following command; `dd if=/dev/urandom of=/tmp/sourceFile bs=1M count=100`.

The original MaidSafe implementation was used to encrypt this file with the CPU clock speed fixed at various levels. This process was repeated 20 times with different random input files, each of the same length, yielding a 95% confidence interval result as to the performance of both implementations.

For a 100 MB random plaintext, the MaidSafe encryption implementation, compiled to a release binary for maximum performance, required a mean time of 10.4 seconds to complete, with a standard deviation of 0.23 seconds. The improved imple-

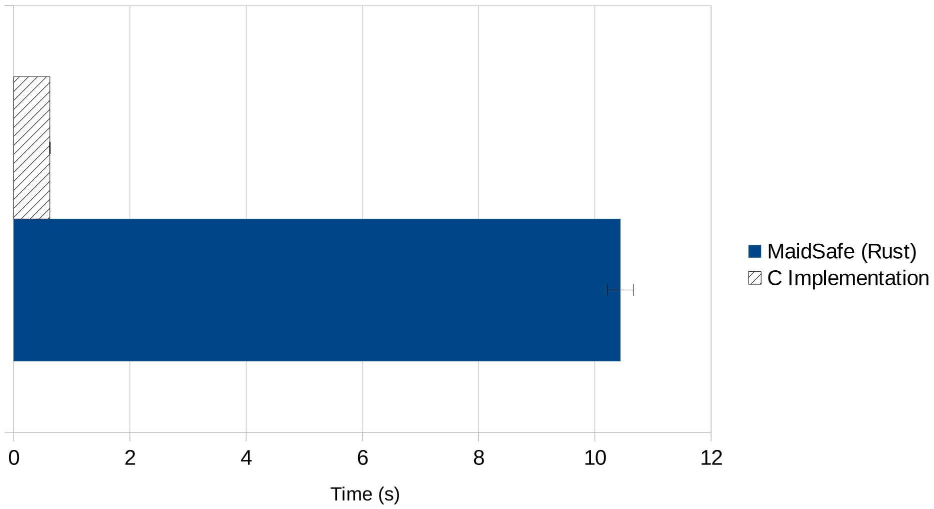


Figure 3.3
Performance comparison between MaidSafe Rust implementation and this C implementation to encrypt 100 MB of data, error bars indicate 1 standard deviation from 20 repetitions

mentation presented here required a mean of 0.62 seconds to complete, with a standard deviation of 0.007 seconds. All timings were the sum of the user-land and kernel-land times measured from the Unix `time` command. This indicated that there were significant performance gains from using a simple C-based implementation. To ensure correct implementation and memory management, the implementation was tested against the Valgrind dynamic analysis tool with all detection enabled, and all tests were passed across a variety of lengths of inputs. Figure 3.3 shows the performance improvement achieved over the MaidSafe implementation.

The security assumptions of this implementation remain the same as with MaidSafe — the Salsa20 nonce was derived from the output of the SHA-512 hash of the data, and this ensures the cipher remains secure, with a unique nonce for each plaintext, provided SHA-512 remains collision resistant. Indeed, this is also an assumption accepted by Bernstein, the author of the cipher, in the upstream reference implementation of Salsa20 [152].

3.2.3 Network Addressing

Within the MaidSafe network, all nodes are assigned a globally unique address, based upon the hash of a public key, which corresponds to a private key held by the user [60]. This address is derived according to Equation 3.1, where A is the resulting node address, $H()$ is the SHA-512 cryptographic hash function, K_{priv} is the randomly generated node RSA private key, and $pub()$ is a function yielding the public key from a given private key. It should be noted that the type of key used could be varied, as could the hashing algorithm. The security of a user's identity is therefore constrained by the security of the process through which their private key was generated, and a strong, verified secure random source should be used for key generation [153].

$$A = H(pub(K_{priv})) \tag{3.1}$$

A cryptographic hashing algorithm, as discussed in Section 2.4, is designed with a number of security properties in mind. These include the uniformity of output distribution, in order that the output is statistically random, provided the input distribution is uniform [59][Section 9.7.1]. If the output of the function $H()$ is taken to be uniform, the addresses of nodes throughout the network should also be uniform. For this reason, the developers of MaidSafe have made the assumption that, in light of the uniformity of the SHA-512 hash function, it would be computationally infeasible to carry out a pre-image attack, i.e. an attack resulting in the selection of a particular node address. Indeed, to do so would require a two-step process — firstly, a public key would need to be found, such that $H(K_{pub})$ yielded the desired address. Secondly, a private key would have to be found, such that $pub(K_{priv})$ yielded the desired public key. Otherwise, it would not be possible for the attacker to assume this identity on the network, as they would be unable to authenticate using the key [64].

Each node on the MaidSafe network is managed by other vaults on the network, and thus a system of mutual management is in place, where all vaults are equally responsible for the health of the network. Node managers are selected by their logical location in the network — a node's 4 closest vaults will act as its managers, with distance measured through the bitwise XOR-distance between node addresses, based on the routing distance metric used within the Kademlia DHT [52]. As such, each node will be responsible for the management of some nodes, and those nodes will be responsible for the management of other nodes.

The bitwise XOR distance between two nodes is, as implied by the name, simply calculated by carrying out an exclusive-or logical operation between the binary representation of two different addresses on the network. Due to the nature of the XOR calculation these distances are unique (i.e. if node B is a distance, d from node A, no other node can be at distance d from either node A, or B), since node addresses are globally unique [154]. Table 3.1 illustrates the relative bitwise XOR distance between sequential numbers for the range of single-digit hexadecimal numbers, given node addresses and other SHA-512 hashes are often quoted in hexadecimal format.

Table 3.1
Bitwise XOR Distances From Previous Number

Hex	Decimal	Binary	XOR distance	Hamming distance
0x0	0	0000	N/A	N/A
0x1	1	0001	0001	1
0x2	2	0010	0011	2
0x3	3	0011	0001	1
0x4	4	0100	0111	3
0x5	5	0101	0001	1
0x6	6	0110	0011	2
0x7	7	0111	0001	1
0x8	8	1000	1111	4
0x9	9	1001	0001	1
0xa	10	1010	0011	2
0xb	11	1011	0001	1
0xc	12	1100	0111	3
0xd	13	1101	0001	1
0xe	14	1110	0011	2
0xf	15	1111	0001	1

3.3 Threats Identified

A series of threats against decentralised networks, built around a peer-managed reputation model shall now be considered. These threats were identified within the context of the MaidSafe network, and include both logical (conceptual) flaws, as well as implementational ones with demonstrations. These threats may also be applicable to other systems implementing decentralised management, or indeed storing user data within a DHT-type construct.

Previous work has reviewed the security of peer-to-peer DHTs, and highlighted a number of concerns such as the return of invalid data, the incorrect directing of traffic undergoing routing and the incorrect claiming of responsibility for a value by a node [155]. Indeed, the MaidSafe network has mitigated these concerns through a number of techniques. The return of invalid data is a concern whereby a client may be unable to detect an invalid value being returned to it, in response to a request. To detect this, data may be held at an address corresponding to its hash. This allows any party to validate the data, including the recipient, by ensuring that $H(data) = address$ [61].

There has also been previous work, in addition to that which forms the basis of this thesis, reviewing the MaidSafe network [156]. Indeed, a number of observations are made by Jacob *et. al* around the design and implementation of the MaidSafe trust model and architecture. In their security analysis, they identify that it is assumed that users have a “non-manipulable” address within the DHT. They also highlight that “an attack on this property, i.e. if nodes could select specific positions, would kill the security architecture of MaidSafe completely.” Jacob *et. al*, however, concluded that the network offered “sufficient protection against mass surveillance attacks”, and that defeating the consensus-based management system of MaidSafe would be “very expensive for an attacker”, requiring either a full Sybil attack, or an attacker which “manages to successfully break the hash function to generate IDs close to a desired position” [156].

This chapter shall show that it is not necessary to break the hash function in order to generate IDs close to a desired position within the network, and that an offline pre-computation attack can be carried out efficiently. As a result of this attack being identified and reported to MaidSafe, the size of a manager group was increased from 4 to 32, with the threshold of majority raised from 3 to 28, as observed by Jacob *et. al* [156].

3.3.1 Address Proximity Attack

A novel attack upon the MaidSafe allocation of addresses was identified in the course of this work, in an area of the MaidSafe protcol reviewed by others and stated to be “very expensive for an attacker” to exploit [156]. The process of generating a device address is described in Section 3.2.3, and results in the output of a uniformly distributed address, as the output of a cryptographic hash function. The original security model of MaidSafe was that each node would be managed by 4 other nodes, and that these 4 nodes would be selected based upon their bitwise-XOR proximity to the node’s address. Within MaidSafe, for two nodes, x and y, their distance is given by Equation 3.2. This distance is considered as a vector of bits, of the same length as the address itself.

$$d(x, y) = x \oplus y \tag{3.2}$$

For any inputs x and y, non-negativity is established, since $d(x, y) \geq 0$. Equation 3.3 illustrates the XOR distance is commutative, as the XOR function itself is commutative. Equation 3.4 demonstrates the triangle equality is also established [52], where distances are considered as vectors of bits. In the equations below, $\vec{a}$ denotes the vector of bits corresponding to the address of a.

$$d(\vec{x}, \vec{y}) = \vec{x} \oplus \vec{y} = d(\vec{y}, \vec{x}) = d(\vec{y}, \vec{x}) \tag{3.3}$$

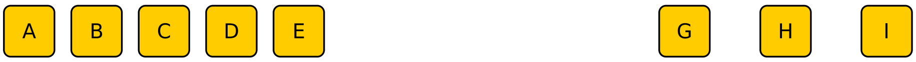


Figure 3.4
Illustration of unidirectionality of bitwise XOR node closeness

$$d(x_i, y_i) + d(y_i, z_i) = x_i \oplus y_i + y_i \oplus z_i = x_i \oplus z_i = d(x_i, z_i) \forall i \tag{3.4}$$

For any given address in the hashspace, x, there exists for a given distance, D, only one other address, y, for which $d(x, y) = D$. This is due to the singularity of the bitwise XOR function, such that for one fixed input, no second input can yield the same output. By way of example, if one node were to have a binary address (shortened for clarity) of $a = 00001111$, and another node were located at $b = 00101110$, then these two nodes have an XOR distance of $a \oplus b = 00100001$. By looking at the least significant bit of this result, it is clear that there cannot exist any other address with LSB of z in the binary number space, such that $1 \oplus z = 1$, other than the previous value of $z = 0$. Therefore, address distance is unidirectional.

From the perspective of a fixed node, there can therefore never be more than one node at any given distance from it. There is therefore no possibility of collision in node addresses based on distance, short of a full SHA-512 hash collision, which should be mitigated by a properly designed cryptographic hashing algorithm, as discussed in Section 2.4. The unidirectionality of the distance metric therefore means that each node may have 4 distinct, unambiguous closest nodes, which may act as managers for the node [156].

3.3.1.1 Distance vs. Closeness

The distance between two nodes is defined, as discussed above, as the bitwise XOR of the two node addresses. Closeness is not in itself defined mathematically as a metric - instead, it is used to refer to the 4 nodes with lowest distances from a given point - these nodes are the 4 closest nodes.

Bitwise XOR node closeness is unidirectional, since one node can be a close neighbour to another node, while the other node has other closer neighbours. This is illustrated in Figure 3.4, where node E will find that its closest 4 nodes are A, B, C and D. In contrast, node G will find that its closest 4 nodes are D, E, H and I. As a result, while the distance between two nodes is symmetric, where $d(E, G) = d(G, E)$, the closeness is not symmetric [52].

Within this example, node E will find that its closest 4 nodes are A, B, C and D, yet node G will find that its closest 4 nodes are D, E, H and I. Therefore, while the distance between two nodes is symmetric, where $d(E, G) = d(G, E)$, nodes are not necessarily mutually close.

3.3.1.2 Vulnerability to Attack

The MaidSafe network makes some assumptions as to the security of SHA512 hashes [156]. Firstly, it assumes the hash output is uniform. Uniformity is desirable, as uniformity is designed to hinder an attacker deliberately positioning themselves towards a particular area of the network. Any kind of bias in the hash algorithm would reduce the difficulty of surrounding a given node. For a truly uniform hash, the mean distance between each node on the network should be approximately equal, as the probability of being placed in any area of the network is equally likely.

The MaidSafe implementation, however, is vulnerable to attack, as a result of incorrect assumptions made as to the security of cryptographic hashing. To demonstrate this, a successful attack must first be defined, based upon MaidSafe's implementation of node management. By considering a successful node insertion attack as one in which a pre-selected address on the MaidSafe network is able to have a new, close, node introduced to it, such that this new node becomes one of the existing node's managers, it is clear such an attack may compromise the MaidSafe self-management model of security [156].

For a hash space of H ($2^5 12$ in this case), and N uniformly distributed nodes, the mean distance between nodes in such a network is given by Equation 3.5. By way of example, for 5 nodes in an address space of 5, the mean distance between nodes would be 1.

$$\bar{d} = \frac{H}{N} = \frac{2^{512}}{N} \tag{3.5}$$

Within such a large address space ($2^{512} \approx 1.34 \times 10^{154}$), the mean distance between nodes is significant.

At this point, the weakness in the MaidSafe implementation becomes clear — it made the assumption that it the uniformity of a cryptographic hash algorithm meant it was computationally infeasible to be able to craft an input which would result in a node being placed at a given location within the network. While the regular property of irreversibility of a cryptographic hashing algorithm does aim to offer this assurance, it does so for targeted outputs. Therefore, the property of pre-image resistance aims to

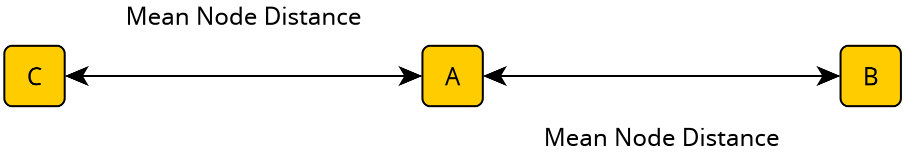


Figure 3.5
Attack space of one address

make it computationally difficult to determine an input which will produce an output of a specific value. This does not, however, guarantee that it will be difficult to produce an output within a given range, which is the property MaidSafe assumed was present within a cryptographic hash algorithm.

If an attacker is able to place a node within the mean distance between nodes, then on average, this node would become the closest node to the existing node. As the closest node to an existing node, the new node would become a manager of the existing node. Indeed, the attacker would still become a manager of the existing node if the attack resulted in the new node having an address closer than the 4th closest node prior to the attack, although for ease of demonstration this attack shall firstly be demonstrated for the scenario of becoming the closest node.

Given a mean distance between nodes of $\bar{d}$, an attacking node will become the closest neighbour if its resulting address falls within $\frac{2\bar{d}}{H}$% of the overall network address space, which is the same as the output space of the cryptographic hash function. This is because to become a close node, it is simply necessary to be closer to the target node than either of the two adjacent nodes. This is shown in Figure 3.5.

For a network of N nodes, it is therefore possible to become closer than the mean inter-node distance in a relatively small number of attempts. As node addresses are uniformly distributed, the probability of a generated address falling within the desired region is 1 in $\frac{H}{2D}$. This is because it is possible for the node to reside on either side of the target, therefore the denomenator of the equation must be multiplied by 2. For a network of 1 million nodes, this means the probability of generating a neighbour for a desired node is shown in Equation 3.6. This can be intuitively validated, since for a network of 1 million nodes, the probability of an attack reaching between any two nodes is approximately 1 in 1 million.

$$P_{comp} = \frac{2 \times (\frac{2^{512}}{1 \times 10^6})}{2^{512}} = \frac{1}{1 \times 10^6} \tag{3.6}$$

Therefore, the difficulty of positioning an attack node within the mean inter-node

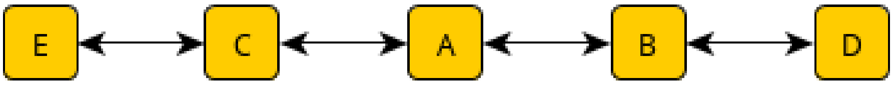


Figure 3.6
Attack against node with 4 neighbours

distance of any given node is entirely linear with the number of nodes on the network. With every attempt at generating an address having a constant probability of success, and each attempt being independent of previous attempts, a successful attack would take place, on average, after half this number of attempts. Therefore, for a network of 1 million nodes, an attacker could introduce its own closest node to any target in half a million attempts, on average.

3.3.1.3 Compromising Further Nodes

These calculations demonstrate the probability of introducing a single malicious node to the network, within the mean inter-node distance for a given network size. Given the definition of closeness on the network, this attack is already considering the definition of closeness, by being nearer to the target than any other adjacent nodes. As these calculations were carried out to be closer than the closest (on average) existing node, which would be the mean inter-node distance, the difficulty of an attack which simply aimed to introduce a new node to replace one of the closest 4 nodes would be easier, given the wider attack space.

For uniformly distributed nodes, Figure 3.6 shows the wider range of attack possible in this case - since each node is statistically uniform in address position, the mean distance between them is, on average, the same. To become one of the 4 closest nodes to the target address, it is now only necessary to introduce the malicious node closer than the current 4th closest node. In this case, where $\bar{d}$ is the mean distance between nodes, this gives the attacker a range of addresses of, on average, $4D$ in size.

With this increased attack target, a node will become one of the 4 closest nodes, on average, if it is able to reach this area of $4D$ nodes. This requires, on average, a quarter of the number of attempts as previously, therefore for a network of 1 million nodes, an attacker could, on average, place a single new node as one of the closest 4 nodes of a given target within 125,000 attempts.

3.3.1.4 Attack Scalability

Operations on the MaidSafe network are validated using a consensus of the 4 closest nodes to the node requesting the operation. A consensus threshold of 3 from 4 is re-

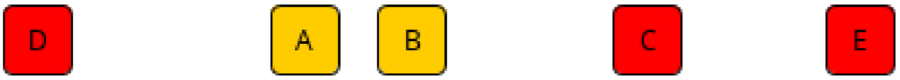


Figure 3.7
Node A compromised by 3 malicious nodes, despite close legitimate node B

quired, in order to approve an action on the network. This means that a threshold of 2 from 4 is also sufficient to carry out a denial of service attack against a user, since an attacker controlling 2 closest neighbours is able to block a consensus from being reached. For the remainder of this section, however, it shall be assumed that the attacker wishes to carry out a complete attack with at least 3 closest neighbours — this allows a false consensus to be reached, therefore potentially overriding the actions of the user under attack, since the 3 attacker-controlled nodes may collude to approve actions.

The effective difficulty of compromising at least 3 of the 4 closest nodes does not significantly increase with each successive malicious neighbour added to the network — assuming that no other nodes join or leave the network while an attack is taking place, the network size will only increase by 1 node following each attack. The proximity of legitimate nodes to a given node should also not typically significantly alter the difficulty of this attack, although it is worth noting that since node distribution is probabilistic and uniform, this is never guaranteed.

By way of example, if a node has one legitimate neighbour clustered very close to it, it is not necessary to be closer than this node - it is merely necessary that the malicious nodes be located closer than the closest adjacent legitimate node. This is demonstrated in Figure 3.7.

Since this attack is based on positioning the node within an incredibly large region of the address space, the probability of achieving this with a relatively small number of attempts is the issue here. The difficulty of positioning 2 malicious nodes around a target is, on average, only twice the difficulty of placing one - this makes the attack linearly scalable. This can be justified by returning to Equation 3.6, and noting that to obtain 2 adjacent malicious nodes, the process needs to be completed only twice — these calculations were made assuming a worst-case scenario, where the attacker wished to be closer to the target node than any other node would be, on average.

3.3.1.5 Scalability

To measure the ability of this attack to scale with a growing network, the number of bits of security provided by the uniform hash shall be considered. That is, while MaidSafe uses the SHA-512 hash to compute addresses, this hash is effectively being truncated for the purpose of closeness assessment, given the significant orders of magnitude of difference between the available address space, and the number of users of the network, as descriped in Section 3.3.1.2. If the number of significant bits of a given address is defined as the number of bits required to uniquely identify a given node, on average, assuming a uniform distribution of nodes in the network, an address offers only $\log_2 N$ bits of security, for a network of N nodes.

Due to this logarithmic relationship, doubling the number of nodes on the network will only add 1 extra significant bit to the hash. For a network of 1 million nodes, the SHA-512 hash is therefore effectively truncated to $\log_2(1 \times 10^6) = 20$ significant bits. By way of comparison, the Bitcoin cryptocurrency uses partially truncated hashes as a proof-of-work scheme, whereby block hashes must be below a given value. For block 389405 [3], a prefix (i.e. truncation) of 69 bits was required, and this task is completed around once every ten minutes as part of the mining process [126]. Therefore, an attack to surround a given user in the MaidSafe network would require significantly less work than successfully mining a Bitcoin. By way of comparison, the difficulty of generating an input, hashing to the address of a node closer than the mean inter-node distance for the network $\frac{2^{69}}{2^{20}} = 2^{49}$ times less computationally intensive, for a network of 1 million nodes. Indeed, even with a network of a 1 billion nodes, this attack remains significantly less difficult than mining a Bitcoin block — the effective hash truncation would result in effective addresses of $\log_2(1 \times 10^9) = 30$ significant bits, still far below the difficulty of carrying out Bitcoin mining, indicating the feasibility of such an attack. Gaining a series of 3 such partial hash collisions would therefore result in having management control over a given network user, and allow for attacker-controlled false consensus to be reached.

3.3.2 Attack Implementation

Until this point, the calculations presented here have been intended to demonstrate that the assumptions placed in cryptographic hashing algorithms by MaidSafe do not offer security against an attacker potentially generating hash inputs which correspond to addresses adjacent to another node. To leverage this design weakness in order to carry out an attack, it is necessary for an attacker to possess the private key, corresponding to the public key which, when hashed, yields one of the four closest addresses to the victim.

Using a regular laptop, with quad-core Intel Core i7 4750HQ CPU, it was found to be possible to generate 1000 RSA keypairs in approximately 62 seconds, using the regular RSA implementation in the Go language [4]. Since this was a relatively slow

[3] https://blockchain.info/block-height/389405

[4] https://golang.org/pkg/crypto/rsa/#GenerateKey

process, constrained by the processing capabilities of the laptop, a distributed attack implementation was designed.

3.3.2.1 Distributed Attack Architecture

This architecture was designed to emulate a scenario such as a bot-net, whereby a series of diverse computing devices were used to attempt to generate large numbers of private key pairs, and return the addresses and private factors of these keys. A Redis key-value pair cache was used as the storage back-end, and it was exposed to a series of Go worker nodes, each generating as many private keys as possible, and indexing them in the Redis database. A client was also created, allowing an authorised user to query the database based on a partial prefix of the necessary key. To ensure high performance of this querying mechanism, the Redis `SCAN` command was used, which matched based on a partial prefix of the string contents of the hexadecimal representation of the target address. Therefore, to ensure partial matching was possible, where the significant bits of a hash did not necessarily align with byte boundaries of hexadecimal characters, a group of `SCAN` queries were made, allowing for each permissible prefix to be located.

Therefore, given the address of a target on the network, it was possible to carry out a search and determine the address and private keys of the closest pre-computed addresses available within the database.

3.3.2.2 Observations

The intention of this work was not to produce the most efficient possible means of storing the private keys generated during this process, based on the assumption of the low cost of storage — a 500 GB Samsung Evo 850 SSD can be purchased for £112 [5]. Storage of an RSA private key was found to require merely the private exponent D, the two factors of the public modulus, P and Q, and the output of the SHA-512 hash, as an index to locate the key. With access to the private key, the public key itself may be retrieved without being stored, by multiplying P and Q [59, Section 3.3]. Therefore, assuming keys with 2048-bit public modulus were required, a total of $128 + 128 + 64 = 320$ bytes of storage was required to hold each key. A 500 GB drive was therefore capable of theoretically holding, assuming fully efficient storage, around 1.4×10^9 private keys and public key hashes.

Based on the calculations from Section 3.3.1.2, a database of 1.4×10^9 keys would be sufficient to compromise, on average, any user of a network containing of the order of 1 billion nodes, given a 1 billion node network would have an effective hash length of 30 bits, and 2^{30} leads to 1.0×10^9 keys necessary, on average, in order to position a

[5] http://www.amazon.co.uk/gp/product/B00P73B1E4

node between the mean inter-node distance between each node.

Based on the performance of the laptop used for testing, as described in Section 3.3.2, it was clear that generation of keys was the performance constraint in this attack. Since an adversary may be assumed to not care about the security of keys, merely their own ability to control a given node as a manager, an alternative attack approach was implemented as an option — if the security, through primality, of the RSA keys used was not considered as significant by the adversary, RSA keys could also be generated using unsafe candidate primes. Profiling of the performance of key generation indicated that the key generation process could be sped up by a factor of over 100 times, if prime validation was not carried out.

By re-implementing the Go random prime generation function [6] to no longer carry out the `ProbablyPrime` check on the prime candidates, generation of RSA keypairs for use in an attack was significantly accelerated — on the same laptop, 100,000 RSA keys were generated in 47 seconds, and 1000 RSA keys were generated in around 300 milliseconds. This resulted in a speed-up of 131.9 times the original implementation.

3.4 Application of the Birthday Attack

While these results are based upon calculating the difficulty of attacking a selected network node, and achieving a rogue consensus of three attacker-controlled nodes, this is not the only applicable attack on the network. Where an attacker has a motive to cause mischief, or where there is any kind of gain able to be made through gaining a rogue consensus around *any* node on the network, the birthday paradox must be considered [59, Section 9.7.1].

The birthday paradox is defined as the probability that any two people in a given group share the same birthday (i.e. date and month). A naive attempt to calculate the probability of two people sharing a given birthday would be that $P = N \times \frac{1}{365^2} = N \times 7.5 \times 10^{-6}$, for N people in a room. While this is indeed the probability that two individuals, from a group whose birthdays are uniform (i.e. not from a gathering of twins, or similar) share a given selected birthday, this is not the same as any two individuals sharing a birthday. In this scenario, the probabilty is much higher, since a birthday is shared if any member of the group has the same birthday as any other member of the group. Specifically, for a group of only 23 people, paradoxically, the probability of at least 2 members of the group sharing a (non-specified) birthday is 50.7% [59, Section 2.1.5].

Therefore, the birthday paradox highlights that the difficulty of surrounding any

[6] https://golang.org/src/crypto/rand/util.go?#L31

node with 3 malicious neighbours may be lower than that of compromising an individually targeted node. In certain scenarios, where there is value to be gained by an attacker able to form false consensus around a node, there may be a motive to carry out untargeted attacks. Alternatively, an attacker motivated by a desire to vandalise may use an untargeted attack to cause denial of service, either by preventing the generation of a genuine consensus, thus preventing a user making requests to the network, or by erasing a user's data.

3.4.1 Potential Attack Applications

The ability to generate a false consensus on the network was identified to be a significant threat for a number of reasons. Putting aside simple denial of service style attacks, which are a constant and ongoing feature of the modern internet [157], rogue consensus can be abused in a number of additional ways. For example, a malicious user may use a false consensus to gain control of a user's data atlas, the section of encrypted data necessary for a user to gain access to this network. By either blocking access to this data, or by generating a false request to delete or overwrite it, it would be possible for an attacker gaining a false consensus to firstly retrieve the data atlas, and secondly to initiate a deletion of the data atlas. At this point, the user would be unable to re-connect to the network, and the attacker would be in a position to ransom a user's access to data back to them.

While similar attacks could be carried out on any other piece of data within the network, since the closest 4 neighbours are used to reach consensus on requests initiated by a user, gaining the ability to create this request to overwrite or remove the data atlas would result in the maximum damage (and therefore most likely pay-out from the victim to gain access to their own data). This attack could be mitigated by users holding local backup copies of their data atlas, in order to re-gain access to the keys needed to connect to the network, and re-locate their data, but this would be somewhat defeating the purpose of a secure, decentralised storage network like MaidSafe, which is designed to free them from needing to store and backup keys offline.

This attack was documented and reported to MaidSafe, and as a result of these findings, the decision was taken to design all data on the network to be undeletable, and ensure that other nodes on the network would reject deletion requests against data. It was recommended that every block of data on the network should be signed by the user's key itself, and that this signature (and the corresponding public key needed to verify it) should be visible and accessible as part of the data. This ensures that even a false consensus cannot alter data generated by a user — without access to the same signing key, the group of nodes holding the data would reject the update as invalid. To carry out this attack successfully, it would therefore be necessary to attack each individual copy of the data atlas, and gain consensus around the addresses holding it, then unilaterally replace the data without informing the rest of the network.

Despite the MaidSafe network being designed to protect details of requests being made against outside adversaries, by encrypting messages exchanged between nodes and their neighbours, it is worth noting that the data atlas may easily be identified by these neighbours, as well as by any other node able to observe the requests being issued, such as the nodes receiving the requests. Traffic analysis allows for the inference of the address of a user's data atlas — the address of the atlas, per the design of the MaidSafe network, remains constant, based upon the derivation of a network address from the user's login credentials [64]. Therefore, the first request made by a user upon connection to the network must be retrieval of their data atlas, in order to gain access to their identity keys and then access their account, making it possible for a malicious attacker to identify the address of the data atlas to compromise.

Mitigation of this attack is not trivial — storage of offline backups of the data atlas was one approach considered, although it would result in an inconsistent view of a user's data if multiple clients were used. Since the data atlas acts as a pointer towards the user's data map, it is necessary for updates to the data atlas to be propogated, in order to gain access to newly added data. For this reason, offline backups do not present a solution to the problem of ensuring the data atlas is correctly held on the network. Instead, this attack is better mitigated by re-considering the threat model of the data atlas, since its loss or corruption would result in the loss of access to all data within the user's account.

3.5 Manager Verification

Much of the MaidSafe security model is built around every node having managers, with these managers being the four closest nodes to the node in question, at the time the operation is carried out. As shown in Section 3.3.1.1, the definition of proximity within an XOR network is unidirectional, meaning that while one of the four closest nodes to A nay be B, this does not imply or indicate that A is one of the four closest nodes to B. This means that, in the absence of an omniscient view of the network, which is not achievable in a decentralised network, due to the inherent lack of central authority with knowledge of every node's address, it is not possible for nodes to generate a map of the network.

Therefore, while a node itself is able to determine its managers, by looking at which nodes are locally closest to it, there is no guarantee that these managers may be able to determine this. Another node may believe that it should be a manager of the node, but there may be other nodes which are closer. Previous work analysing the network has stated as a security assumption that users assume "the view on the network is equal for all nodes" [156], but given the uni-directional nature of closeness within the network, this cannot be guaranteed without an omniscient view of the network, requiring it to be centralised.

This limitation poses a potential concern for the MaidSafe network — while it is possible to carry out theoretical analysis of the network from an omniscient perspective, and demonstrate the difficulty of becoming managers of a given node, without an omniscient entity on the network, the potential for such an attack cannot be ruled out, since nodes can only rely on the nodes they themselves can detect. If an attacker were able to disrupt communications or segment the network, such that a node could be tricked into believing that colluding nodes were its closest neighbours, it could be convinced to join an attacker-controlled group.

Per the Kademlia network design [52], close nodes can be discovered through a request which is propagated through the DHT. When a request to find close nodes is made, a DHT node issues a propagating request with the target address it seeks. Nodes progressively closer to the target address will respond with the closest nodes they know to the target address, resulting in an eventual convergence on the closest nodes to the target address. Since this query is carried out through multiple nodes, and nodes know more nodes in their close address-space, as discussed in Section 2.3, the final nodes located will be the closest responding nodes to the target address.

3.6 Chunk Information Holders

As discussed in the MaidSafe network overview in Section 2.5.2, data on the MaidSafe network is held in chunks, which are distributed throughout the network based on the hash of the encrypted chunk. At the logical area of the address-space, where the chunk's hash is located, the closest 4 nodes are selected as "chunk information holders". These nodes are responsible for managing the chunk in question — their tasks include requesting other nodes hold the actual data, and then verifying this data in future. These nodes also handle updates of mutable data, by verifying the request, as well as handling the garbage collection of data, by requesting the deletion of data when it is no longer required.

This puts chunk information holders in a highly privileged position, within the context of the MaidSafe network. The 4 nodes are responsible for selecting nodes to hold the data — 4 copies of each chunk are held on the request of the chunk information holders, and the nodes holding the chunks are then managed by their own managers (their 4 closest nodes). This architecture gives rise to a number of vulnerabilities, on account of the unbalanced privileged access given to the chunk information holders. While the issue of selection of chunk holders was discussed previously from the perspective of the lack of determinism or randomness in the allocation, there was no security implication highlighted [156]. A number of practical security implications of this shall be demonstrated.

3.6.1 Data Distribution Authority

The first weakness identified was that while chunk information holders are selected deterministically and uniformly, as a result of the cryptographic hash of the chunk, the nodes used to hold the actual chunk data itself are selected entirely arbitrarily by the chunk information holders. This means that, in the event of malicious nodes being introduced to the network, they may rapidly begin to influence the distribution of data on the network. A group of rogue chunk information holders may issue chunks to a group of nodes under their collusion, if they wish to gain control of the stored copies of data. This would potentially allow these chunk information holders to refuse access to data, on request. Since the storage of the resulting chunks is arbitrary, and not auditable by a third party through deterministic locating of data, it is not possible to determine wrongdoing in the actions of chunk information holders. If they were to request data be stored only on cooperating nodes, it would be possible for an attacker to gain control over all copies of the chunks they act as information holder for, violating the availability guarantees by the MaidSafe network.

3.6.2 False Data Mutation

Data mutation requests are generated by the owner of a given chunk of mutable data, and the request is validated by their four closest neighbours, per the standard consensus-building approach. Once consensus on the action is approved, the action is relayed to the chunk information holders of the chunk in question. The chunk information holders validate the consensus, ensure the data is not marked as immutable, then request the individual chunk holders update their copy of the chunk to the new version. In the event of either a node's managers being compromised, or a chunk information holder being compromised, it would be possible for data to be falsely mutated. As a result of this, an appropriate mitigation would be to ensure that all mutable data is signed, and that mutation requests are signed with the same public key as the original data. This prevents a compromised chain of trust between the node's managers, and the chunk information holders, from being used to falsely mutate user data, causing data loss. This is naturally only a concern for mutable data, since it cannot inherently be validated against its address in the same manner that immutable data can be. Therefore, by requiring the nodes holding chunks to validate the signature of mutated data, it is possible to ensure that the data is genuinely produced by the same user who initially stored it, thereby preventing vandalism.

To demonstrate this, an extension of this solution has been implemented in Section 3.10, to allow for transferable-ownership of signed data, while ensuring that a chain of validity is preserved, allowing for a node to re-join the network after any period of inactivity, and still verify the legitimacy of any update, without allowing an attacker to downgrade data to maliciously alter ownership data.

3.6.3 False Data Removal

Since nodes holding chunk data are accountable only to the chunk information holders, it is the responsibility of chunk information holders within the MaidSafe network to identify when data is no longer required, and organise the removal of the data. To ascertain whether or not a chunk is required by a user, without maintaining a list of users requiring a given chunk, the MaidSafe network carries out a repeated XOR process, whereby a chunk's watch list is XOR'd with the identity of a node requesting a chunk be retained. Therefore, for a chunk requested by two nodes, the chunk watch list will hold the value of $A \oplus B$, for two nodes, addresses A and B.

From the perspective of a chunk holder, however, the only way to verify the authenticity of a request is to ensure it came from the chunk information holders, which are the 4 closest nodes to the chunk address. Since the request itself is not directly relayed by the chunk information holders to the chunk holders, the chunk holders can only verify the deletion request based on the details they receive from the chunk information holders. As the chunk information holders process all requests to delete data, they therefore have the ability to follow the value of the chunk watch list. To determine when the last chunk watcher has been removed, the list is XOR'd with the watcher identity unsubscribing from the chunk, therefore meaning that when all watchers have chosen to remove the file, the watch list will be left as an all-zero value. The chunk information holders may therefore indicate to the chunk holders that the data is no longer required, by simply returning the current value of the chunk watch list, since for watch list value L, $L \oplus L = 0$, therefore resulting in an empty watch list, resulting in the removal of the chunk data, even though it may be required by nodes.

Since the chunk information holders are responsible for the control of the 4 copies of the chunk held on the network, it is possible for them to cause data loss for users, by exerting their control over all copies of the data on the network. As a result of our reporting of this issue to MaidSafe, it was agreed that the best mitigation against attacks such as these would be to make all data on the network undeletable, until such a time as a suitable mitigation was identified.

3.6.4 Mitigation and Analysis

When the original threat model of the MaidSafe network was carried out, attacks like this resulting in the false deletion of data were not directly considered, as the network's reputation system was assumed to provide sufficient guarantees against the growth of malicious nodes in the network — the reputation network was designed to ensure that data was stored based on availability and capability of nodes to hold the data in question, with data removed from low-reputation parties [25]. In the event of collusion of a number of entities, their reputation need not necessarily suffer — while if nodes were to generate their own commands falsely, or refuse legitimate commands, these would result in detectable refusal to follow the rules of the network, this is not the case for all attacks. If malicious chunk information holders were to only attempt to target chunks for which collaborating malicious nodes held a majority of consensus, they could carry out attacks such as those described above, without fear of reprisals, since only the chunk information holders themselves hold the information necessary to determine whether a file should be retained, and its content.

In order to mitigate this risk, it is necessary to ensure that chunk information holders do not hold a privileged position, granting them exclusive access to manage data. By carrying out a small number of changes, it is possible to ensure that chunk information holders cannot directly cause data loss or denial of service without detection. Firstly, the storage locations of chunk data should be deterministically derived from the hash of the chunk data — rather than allowing the chunk information holders to arbitrarily locate chunks, the nodes selected to hold data should be selected based upon a modification of the cryptographic hash of the chunk contents. For example, for a data chunk whose address (and therefore chunk information holders) is A, copies of the chunk may be held by the node closest to address $H_n(A)$, where H_n is the recursive cryptographic hash function, with $n \in \mathbb{Z} \mid n > 0$, for n iterations.

Any member of the network may therefore verify that data is being stored at the correct location, without the chunk information holders having the ability to select colluding nodes to hold data. While this process does not mitigate concerns with regard to the formation of consensus and close groups, as discussed earlier in this chapter, this mitigates the specific risk of a rogue chunk information holder group having considerably more influence than other rogue nodes on the network.

3.7 Dishonest Vault Attack

The dishonest vault attack is explored in further detail here. In this attack, a user can upload more data than they are themselves storing, thus bypassing one of the limitations of the MaidSafe network, whereby users would be restricted to storing only up to the average of other users, unless a node offered more storage space. This serves to balance the needs of users to store data, while also ensuring that malicious users do not fill up every vault on the network with meaningless data, thus preventing others from storing data. To balance this, MaidSafe had proposed that users should only be able to store as much data as they store for other people.

3.7.1 Overview of Attack

The dishonest vault attack is a concern when other mitigations to attacks are considered, such as preventing the deletion of data, to mitigate attacks against chunk information holders, as described in 3.6.3. If data is not able to be removed from the network, to prevent denial of service through data deletion, it is important to ensure that users cannot upload more data to the network than is sustainable, as their data may not be removed after it is stored. Since, from the perspective of members of the MaidSafe network, all data is opaque to the holder, it is not possible for it to be filtered or otherwise scrutinised; such decisions can only be made based on rules such as the node of origin, and its reputation.

This attack involves a user generating their own vault, and offering to store significant quantities of data for other users. Note that this attack may be combined with the storage hand-off attack, considered in Section 3.9, for enhanced efficacy. After offering significant storage to the network, the user may store data per the network rules, then revoke their storage offered.

3.7.2 Analysis of Impact

Since data was proposed to not be deletable on the network, the user has effectively unlimited storage. While their account reputation may be decreased as a result of this, since any user may retrieve any data from the network, to allow for login to take place, it would not be possible to deny the user access to the data, even if they were to retrieve it from a new account.

While this would require a custom client implementation, in order to use a different account key to decrypt data, this cannot be prevented by the network, since retrieval of data is separate from decryption of data, and it is never necessary to prove ownership of data to the network, or it would not be possible for users to take part in verification of data integrity or other network housekeeping tasks to ensure reputation scores are held correctly.

3.7.3 Mitigation of Attack

In order to mitigate this attack, it would be possible to adjust the storage limitation algorithm, such that clients may only store as much data on the network as they have already been verified to hold. While this cannot prevent a user from following the protocol to gain storage, holding data, before later removing it, that is a consideration more towards the deletion or expiry of data, rather than towards the validation of storage — nothing other than reputation can force that a user keep their vault online, without offering opportunities for loss of data or denial of service.

This does, however, raise a new challenge, namely around how a user may join the network and establish credibility — without a reputation, a user will struggle to find nodes willing to store data on their system, therefore preventing users from storing data on the network and thus gaining reputation by following the rules of the network. To prevent this, a proof-of-storage mechanism is proposed in Chapter 4, which would allow a user to store verifiable data, and allow a third party to verify this, without requiring others to entrust the vault with their data. It is worth noting that the MaidSafe network does have design provisions for this scenario, since the duplicate copies of chunks ensure that nodes are not holding the sole copy of a given chunk on the network. Nonetheless, a means of proving the storage of data offers an approach through which users may gain reputation on the network following their entry, without others being required to trust them.

3.8 Incorrect Proof of Storage Implementation

As introduced in Section 2.6.1, the challenge of resource management is significant within a decentralised network, given the lack of a central entity to manage and enforce storage quotas. One of the approaches considered in previous works includes proof of storage, whereby actors on a network provide a proof to others that they are storing data. This proof may be validated by other parties, in order to validate the veracity of the claim of storage on the network.

One of the limitations of a proof of storage is that it often necessitates the verifier to also hold the data being validated. This makes validation of proofs of storage difficult to scale, and thus disincentivises others from validating claims of storage being provided, as they would also have to store the data themselves.

3.8.1 MaidSafe Implementation

The current MaidSafe proposal for proof of storage [7] is based around a cryptographic hashing technique, whereby a node may prove to a group of verifying nodes that it holds a given piece of data by returning a response to a challenge.

The protocol currently designed by MaidSafe requires that the verifiers present the node with a challenge string. This string is appended to the original data chunk whose presence is being proved, and the response is given to the verifiers. They may verify this by carrying out the same operation. During review, it was established that this construct is vulnerable to a length extension attack, whereby an attacker without storing the data could return a correct response to a challenge.

In a length extension attack, which is applicable to a wide variety of hashing algorithms, including the MD5, SHA-1 and SHA-2 families [158], it is possible for an adversary to compute the hash of 2 concatenated messages, such as $H(a||b)$, without knowledge of both a and b, provided suitable padding is allowed for. This could remove the requirement for the user to store the contents of a, and thus negating the premise of the storage proof. Indeed, given the MaidSafe network identifes ciphertext chunks by their hashes, as discussed in Section 2.5.1, there is no requirement for a malicious prover to even store $H(a)$, since they will be provided this to carry out the proof. Previous work has produced software which will generate such extended hashes [8].

A direct and immediate mitigation for this weakness would be to use a proper keyed hash construct, such as an HMAC function, which is designed to be resistant to such length extension attacks when combining two parameters [9].

3.8.2 Implementation of a High Performance Proof of Storage System

The concept of proof of retrievability was introduced as a way to allow a server to prove to a client that it can retrieve a given file, without requiring the transmission of the full file itself. Therefore, the client may be satisfied that the server holds the data, and that it is not corrupted, truncated or tampered with, and that it would be retrievable if required [77]. The proof of retrieval was introduced as a special case of a proof of knowledge, designed to cover a large stream of data, such a file or chunk.

One challenge of implementing a proof of storage protocol is that of ensuring efficiency for the verifier — if the verifier is required to store the full file, in order to validate the verification response, there is little advantage for the verifier, as they need to ensure they retain all their data independently. Therefore, it is necessary to ensure that, at least

[7] https://github.com/maidsafe/SystemDocs/blob/master/en/system_components/proof_of_resources.md

[8] https://github.com/bwall/HashPump

[9] https://rdist.root.org/2009/10/29/stop-using-unsafe-keyed-hashes-use-hmac/

for the purpose of verifying a new node's credibility in storing data, it is possible for verification to take place without the verifier retaining the same quantity of data as being held by the new node.

One approach is to generate a finite series of challenge-response verification questions, and pre-compute the answers to these. By way of example, to request verification of a segment of a chunk, from byte 100 to byte 200, the verifier may pre-compute i keyed hashes ($HMAC(key, data)$) of that range of data, for a series of i keys (K_i), with a given per-challenge nonce N, i.e. $V_i = HMAC(rnd\_key_i, chunk[100 : 200]||nonce_i)$. A keyed hash should be used to prevent length extension attacks being carried out on the hash, since an implementation may append the nonce to the hash of the data. Provided that the node being verified cannot predict the challenges used, it is necessary for it to retain the full data, in order to be able to present the keyed hash of a given range of data. The verifying node would have to store a number of pre-computed challenges, for different ranges of data, and with different nonces and HMAC keys, in order to ensure that the node being verified is not likely to pass verification without holding the full file. The process of verification is therefore somewhat complex for the verifier as it requires them to pre-compute sufficient challenges to use in future.

An alternative approach would be for the verifying node to generate synthetic data for the purpose of establishing credibility by a verifier. By using a random-access stream cipher, such as Salsa20, it is possible to generate an unpredictable pseudo-random sequence securely, and store only the seed parameters needed to re-initialise it [151]. By storing only a 32 byte key and 8 byte nonce, it is possible to generate the same output in a repeatable manner. This allows the verifier to store only 40 bytes, and generate, in constant time, any arbitrary position of the output, since the output is entirely a function of the position within the output [159].

A verifier may therefore provide a new node wishing to gain reputation with an arbitrary block of random data from the output of the stream cipher, and request it be stored. The verifier may, with only 40 bytes of data stored, verify any portion of the contents of the data, by merely requesting that segment of the data, or any function derived thereof. The purpose of this is to permit a node joining the network to *bootstrap* trust by demonstrating its capability to reliably store data, at a point where its reputation may be non-existent, meaning that no user would wish to rely on the node to store data on its behalf.

From a security perspective, the best current cryptanalysis of the Salsa20 stream cipher has managed to break 8 rounds [160], and 15-round Salsa20 has been proven to offer 128-bit security against differential cryptanalysis [161], showing that differential cryptanalysis would require more effort than an exhaustive attack against a 128-bit key.

The key properties of a stream cipher are that knowledge of its output should not reveal either the initial state, or the key to the stream cipher, and that the output should be indistinguishable from random data [162]. Since the best cryptanalysis to date against Salsa20/20 has shown no progress towards violating either of these, knowledge of output of the stream cipher would not allow a node to recover the initial stream cipher state, which would have allowed a node proving storage to avoid storing the data in question, by using the same technique as used by the verifier.

Section 4.2 continues this work by applying this concept for the purpose of storage limitation and upgrades, within a decentralised network, and shows how this can be used to form a decentralised contract protocol, allowing mutually untrusting users to form contracts to offer storage, while being externally verifiable.

3.9 Storage Hand-Off Attack

The storage hand-off attack presented here is a means through which a malicious party on the network may falsely gain reputation or credibility, without offering the storage and utility to the network which would ordinarily be required to gain that reputation. Within the MaidSafe proposal, nodes are required to store data they are allocated, in order to gain and retain reputation. By penalising nodes which lose or corrupt data, and by storing data on reliable and well-reputed nodes, data is more likely to be stored in a reliable location on a node with reliable storage and a reliable connection, on account of reputation. The edge-case of a previously reliable node going offline is handled through the inbuilt redundancy of the network.

Since nodes are rewarded through the reputation system for the storage of data, and potentially also through the SafeCoin currency proposed by MaidSafe, there is an incentive for a malicious node wishing to gain reputation to attempt to have the network believe it stores more data than it actually does. Such an attack could allow a user to gain the reputation benefit of storing that data, without having to actually store it reliably.

Within the MaidSafe network, immutable data may be easily integrity-verified, since the hash of a chunk's contents is its address. Therefore, if address A contains data, D, this data may be verified by ensuring that $H(D) = A$. Such a verification process requires that the data be retrieved for verification, although an alternative proof of storage process, not requiring the retrieval of data, was discussed in Section 3.7.

In the storage hand-off attack, it would be possible for a node to be asked to hold a given chunk, and to then discard the chunk, having verified that it was held by 3 other nodes. In the event that the node was required to produce the chunk for verification, the node may retrieve the chunk by making a regular data retrieval request on the network, and then relay the retrieved chunk to the node requesting verification.

Other than by attempting to infer the likelihood of such actions based on request latency, there is no clear solution to this problem within the MaidSafe network's design — it is logically not possible to verify if data is actually stored on a system, rather than stored elsewhere and loaded into memory when required. In this case, elsewhere could be on other nodes on the MaidSafe network.

An approach to prevent storage hand-off from being used by malicious network members to falsify the appearance of offering a large quantity of storage to the network is to require that each node hold a different version of the data. This way, it is not possible to offload the chunk, as the data held is unique. In order to preserve availability in the presence of offline nodes or nodes which depart the network, it is necessary to ensure redundancy remains available. One solution would be to utilise erasure codes across chunks — this would allow for recovery of data in an n-from-m fashion, where any n blocks from m are required to retrieve the data, with a loss of up to $m - n$ blocks permitted without data loss.

An erasure code library such as zfec [163] would offer each chunk holder a unique block of data, which they were required to store. In the event of the loss of a chunk, this would be detected in the data verification process, and another node may be allocated that chunk to be stored, after the node responsible for holding the lost data is absolved of the responsibility and suffers the appropriate reputation decrease as dictated by the network rules. The use of erasue codes across entire files would also increase the resiliency of the MaidSafe proposal, since the loss of all 4 copies of a single chunk results in the loss of access to the file in question.

3.10 Decentralised Transferable-Ownership Mutable Data

As discussed previously, one of the recommended solutions to handling the challenge of falsely mutated data was to carry out signature-based verification of all mutation operations on the network. While immutable data does not require any ownership records, since it cannot be updated or removed after creation, mutable data requires either a consensus-based approach to validation of requests, or a signature-based approach. The former approach has been shown in Section 3.6 to be vulnerable to false consensus attacks, specifically around the chunk information holders although also around the node owning the data.

MaidSafe has proposed to implement a signature-based approach, where data must be signed with the same signature as used by the originator of the data. For example, within the context of the MaidSafe network, a node of address A would sign the original mutable data with the private key K_{priv}, corresponding to $K_{pub} = A$. In order to mutate the data, this requires that the user retains their private key, and it is not compromised. It also precludes a user from transferring data into the control of another user, as would be required in the event of a user believing their keys were compromised, and wishing to fully re-key their account.

Additionally, a number of new opportunities are presented in a network allowing for the transfer of data between users. By allowing for the transfer of ownership of DHT keys, they become something which may be traded or exchanged between users. Given the MaidSafe network has proposed a scheme for friendly-naming of resources based upon hashes in the DHT [164], the ability to securely and verifiably transfer a named resource between users allows for various opportunities, such as the exchange or sale of domain name-like identifiers. Related techniques allowing for mutable data within DHTs have been proposed, although these do not permitt the transfer of ownership of identifiers [63], meaning it is not possible for users to re-key their data.

3.10.1 Challenges of Signature-Based Approaches

One key challenge of a signature-based approach, within a decentralised network, is preventing the rolling back of data. A malicious attacker could potentially roll back a user's data atlas, by initiating a mutation request against the data, using an old copy of it (which was therefore validly signed by the owner). One simple mitigation would be to not store the signature alongside the data; merely the public key owning it, therefore preventing a malicious party from gaining the signature requesting its storage. This would not be ideal, however, as any party with access to the request, such as a user's own managers, the chunk holders, or the chunk information holders, could gain access to the signature in order to replay it. Additionally, this would prevent any other party on the network from validating the integrity and legitimacy of the data being presented, and require anyone retrieving the data to trust the chunk holders to not collude to modify it. Better approaches would permit any such tampering to be detected by any member of the network, as with regular immutable data, using a content-defined address.

An alternative approach would be to use a challenge-response approach, whereby the individual chunk holders require a unique nonce to be included in the signed update message. This once again assumes that the chunk holders would not collude to attack data, and also prevents others from validating the content of the chunk, since the signature would not be verifiable without knowledge of the challenge, and assurance that the challenge was unique and not previously used before.

Any ability for a value to be replayed would allow for a malicious party to reverse a transfer of a key to a new owner, potentially allowing for fraudulent sales to be carried out; once the payment was received, the key could be reverted to the original owner, leaving the new (rightful) owner with neither their money nor the desireable key they had purchased (like a domain name).

One other challenge posed by the signature-based approach is that of nodes which have been offline for a period of time, and which later re-join the network. It is important to ensure that these nodes are not required to blindly trust the assumptions made by the rest of the network, particularly where resources may be exchanged for money, or where ownership of keys is changing. Therefore, it is necessary to ensure that any change in ownership of a key is verifiable by any third party which has been offline for a long period of time, but which was monitoring the key in question. It remains important that during this verification, it is possible for the previously-offline node to ensure that the order of transfers is also correct — if the transfers can be replayed in the wrong order by a malicious party, it could result in the node gaining an incorrect view of the ownership of the key, potentially allowing it to be tricked into recognising incorrect ownership. Throughout the process however, the hash of the address must not be changed, otherwise the utility of being able to pass ownership of a key (an address) in the network would be eliminated.

3.10.2 Formulating an Ownership Structure

To formulate a structure which may represent the ownership of data, and also handle transfers of this ownership, without resulting in a loss of transfer directionality, the previously identified challenges can be used to form a series of requirements. Firstly, all updates to data must be signed by the private key corresponding to the owner of the chunk, in order to ensure that false mutation requests are rejected. Secondly, these updates must contain an order field (for example, a counter), in order to ensure that older signed requests cannot be relayed over newer ones, resulting in a downgrade of the data version held on the network. Thirdly, transfers of ownership should be verifiable, and held in a chain, allowing a party offline for any arbitrary period of time to ensure ownership has been correctly transferred. This means that it is not necessary to retain every previous value of the data; merely to store the requests altering the ownership of the chunk. While one approach would be to build a separate chain of modifications, each held in a separate key, such as that of a Blockchain construction, this would result in the generation of many more keys, presenting an overhead to the network upon requests to change ownership, and also require the retrieval of many more values than would otherwise be required, which may cause delays given the routing latency of a DHT.

The solution presented therefore aims to take the simplest approach which also satisfies these requirements. Therefore, the following specifications are introduced:

- Every update message must be signed by the current owner of the chunk
- Every update message must contain a sequential counter field, which increments monotonically
- The current owner of a chunk must be defined within the chunk's data, adjacent to the chunk's value
- An array, containing previous chunk ownership change approvals, must be contained adjacent to the chunk's value
- All ownership changes must indicate a new owner public key, which must be appended to the ownership change array
- Ownership may be verified by following the chain from the earliest-known owner, and ensuring that the current owner is reached through an uninterrupted chain of ownership transfers

To maximise the use of existing web standards, the JSON Web Token construct was used to represent transfer messages and signed data — JWTs are compact and standardised [165] brief messages, with low overhead, designed for the secure exchange of signed data. A JSON Web Token contains three fields; a header, a body, and a signature. Each is encoded into base64 form for transfer, therefore ensuring the encoding is safe to transfer via URL-encoded or form-encoded data on the web. Each field is concatenated together, separated by a full-stop character. The final field is then a signature, with the algorithm of the signature indicated within the header field of the data. While some implementations of JWT have been found to be vulnerable to manipulation as a result of their trusting of header fields prior to the validation of the signature, this does not pose a risk to implementations which specify the accepted signature schemes [166].

The header of a stored record on the network should contain the information shown in Listing 3.1, and the body data is shown in Listing 3.2.

Listing 3.1

Required JWT header

```
{
  ”id”: ”abcd”,
  ”owner”: ”8ef20e8897be93df4cf521d...”,
  ”version”: ”1”
}
```

Listing 3.2
Required JWT body data

```
{
  // arbitrary data held here
  ”stringVal”: ”Arbitrary data here”,
  ”booleanVal”: false,
  ”integerVal”: 9
}
```

By storing the encoded JWT, it is possible to verify the signature on the token, and thus verify the integrity of a given token upon its retrieval. The header of the JWT presents the current owner of the chunk, therefore allowing verification of ownership update messages.

In the event of issuing an update to this chunk, at the same address, two changes must be made — the version flag must increment to 2, the next un-used version available, and the body should be updated to contain the new data. In the event of an ownership change, the version flag must again increment, and the owner field should be updated to contain the public key of the new owner.

Nodes receiving update messages incorporating a change in the owner field must store such updates — for this reason, they should not specify a message body, since they are not updating the content of the chunk, and the entire JWT must be retained to validate the signature. This would therefore require the storage of previous chunk contents, which may be undesireable in the event of a user wishing to re-key following a suspected compromise or loss of their keys, or potentially following a major advance in cryptography requiring a different algorithm to be used.

Therefore, a node storing a chunk must retain the data shown in in Listing 3.3. Note that a “Transfer JWT” is as defined in Listing 3.4, and may be stored in the transfer history field as received, in base64-encoded format. To validate a transfer chain, the first item in the `transfer_history` array is decoded, and the owner obtained. That owner public key is then used to validate the signature on the second transfer, and the process is repeated for all subsequent transfers in the list.

Listing 3.3
Node chunk record

```
{
  ”id”: ”chunk_address”
  ”owner”: ”current_owner_public_key”,
  ”transfer_history”:
    [
      {Transfer_JWT_1},
      {Transfer_JWT_2}
    ]
}
```

Listing 3.4
JWT ownership transfer data

```
{
  ”id”: ”abcd”,
  // new owner follows
  ”owner”: ”8ef20e8897be93df4cf521d...”,
  ”version”: ”2”
}
{
  // null body
}
// signature by previous owner
```

By storing base64-encoded transfer JWTs, using the minimal scheme detailed above, with truncated JSON keys to reduce the overhead of human-readable keys, a historical ownership message may be represented in as little as 242 bytes — 20 bytes are required to represent the overall JSON structure, 44 bytes for the base64-encoded Ed25519 32-byte public key of the new owner, 2 bytes for the version field, and 88 bytes each, for an un-truncated SHA-512 hash output as the chunk address and the base64-encoded Ed25519 signature. Ed25519 is a high-performance asymmetric signature scheme, notable for its short signatures (64 bytes) and public keys (32 bytes) [167], ideal for minimising the length of each ownership record, while retaining a target of $2^1$28 security, and having a best-known attack of $2^1$40 bit operations of equivalent security.

Therefore, an ownership history may be stored, for n previous owners, with an overhead of 24 bytes for the JSON structure, 88 bytes for the chunk address, and 242 bytes for each record n (with a JSON overhead of 2 bytes per record). This offers a low overhead approach to the storage and representation of an ownership history record, while ensuring that a full audit trail is preserved to satisfy any node as to the legitimacy of ownership changes that occurred while offline. A key with a history of 4 previous owners could therefore have its full ownership record represented in only 1088 bytes.

3.10.3 Threat Modelling

The possible attacks on this construction may be threat modelled — firstly, the possible attackers are identified, and then the types of attack are explored. The assumptions made in each scenario are stated below.

There are three actors to be considered — the current owner, a previous owner, and any third party who has never been an owner of the chunk.

3.10.3.1 Third Parties

Third parties are assumed to not have access to a user's live signing keys, and therefore cannot generate signed update requests which would be accepted. A third party could replay an old signed message, but the counter field of these messages would be below the current version, therefore preventing replayed messages from being accepted. In the absence of valid signing keys, however, or holding a pre-signed update request which had not yet been submitted to the network, a third party cannot generate an acceptable ownership update message.

3.10.3.2 Previous Owner

A previous owner is a special case of a third-party, where the party has previously held the valid owner key for a chunk, but has since submitted an ownership change to the network. Since a previous owner is not detailed as the current owner, and is only referenced in the historical ownership change record, there is no difference between a previous owner and a third party, from the perspective of forging ownership updates.

In the special case of a number of outdated nodes entering the network, there is a risk that a malicious previous owner could relay a false chain of updates to this node, in the event that the outdated node still was of the view that the previous owner controlled the data. This would be resolved through consensus on the network, since the data held by the node would be out of line with that held by the other nodes holding the same replicated data. This attack is somewhat equivalent to the risk of a parallel side-chain forming in a Blockchain-like construction, but is mitigated when the node itself identifies the data it holds is no longer in line with its peers. Since a decentralised network like MaidSafe holds data in multiple locations, an attempt to carry out this attack would result in inconsistent data on the outdated node, which could verify the ownership history was different from the remainder of the network, and report the attack to other nodes, while updating to the network's view of ownership.

3.10.3.3 Current Owner

The current owner of a chunk is able to produce a signed message which will allow for ownership to be updated. The process of transferring ownership is ordinarily irreversible — after a third party verifies the new owner record is present, perhaps from more than one of the chunk holders to mitigate against a single rogue chunk holder, the former owner is no longer the owner of the chunk, and therefore cannot carry out mutations on the chunk. This prevents the reversal of the transfer, since the transfer would need to be signed by the recipient of the chunk.

This highlights one of the risks identified earlier; namely that of the risk posed by chunk information holders selecting the nodes to hold the final chunks. In this scenario, it is possible for a rogue chunk information holder group to select colluding chunk holders. For this reason, it is important to ensure the mitigations detailed in Section 3.6.4 are applied, so that chunk holders are deterministically selected from the chunk address, rather than arbitrarily selected.

Provided the current owner cannot collude with the chunk information holder to select colluding chunk holders, the current owner will therefore not be able to reverse the ownership transfer, as the chunk holders would then reject the second transfer due to not being signed by the new owner.

3.10.3.4 Evaluation

This approach offers security against previous owners changing the ownership records — other nodes will reject their attempts to present a new signed ownership record, since previous owners are no longer listed as the current owner. Legitimate chunk holders will reject replay attempts, since a replay would result in the decrement of the chunk's sequential counter. Since the signature of the current owner is validated for all value updates, or ownership changes, a third party cannot satisfy legitimate chunk holders that a mutation request is valid.

It is important to ensure that when signature verification is carried out, it is done in the correct order — the security properties of a signature are such that it is possible to verify a message is not modified since being signed by a given, known public key [59, Section 1.8.3]. These security properties do not extend to allowing for the verification that a given message originated from a given public key, as was shown by Koblitz and Menezes in their presentation of a practical *Duplicate Signature Key Selection* attack [168]. In this attack, it is shown that for a given signature and message, a non-trivial public key can be generated, such that the signature will validate against the public key, and the attacker holds the corresponding private key. It is therefore essential to ensure in an implementation such as this, that the signature is validated against a known-correct public key.

For the case of transfer of ownership, this means that a data mutation or ownership transfer request should include a header (within the signature), detailing the address of data to be mutated on the network. At this point, the declared address should not be regarded as trusted, although the corresponding public key owning that key should be retrieved from the network. This indicates the current owner public key to the verifier of the request. The signature on the received message should then be validated against the public key retrieved from the existing record on the network, and specifically not from any public key in the request or any other location. Following this verification, it can be ascertained that, for the data chunk of the stated address, the signed request originates from the true owner of the chunk. If the wrong address was claimed, an incorrect public key would be retrieved from the existing network record, and therefore the incorrect public key would be used for validation, ensuring that validation fails.

3.11 Conclusions

A security evaluation of the MaidSafe network was carried out, in order to investigate the security of various components of the network. A number of weaknesses were identified in the original implementation of the decentralised management protocol. These weaknesses were disclosed to MaidSafe, in order that they may be fed into the development of their network.

First, it has been shown that allocating managers based upon address-based proximity or closeness is inadequate, and that the difficulty of carrying out an attack, thus gaining control over manager nodes, is low. Specifically, it was shown that, for a network of 1 million nodes, the difficulty of placing a node in a location which is below the mean inter-node distance was 2^{49} times less computationally intensive than the process of mining a single Bitcoin block; a process which takes place approximately every ten minutes. This highlights the difference in order of magnitude of the processes, and that if a large-scale attack was suitably motivated, it could be carried out.

Second, the challenge of correctly identifying which nodes are another node's managers has been highlighted — while related work has assumed that "the view of the network is equal for all nodes" [156], this is not necessarily achievable within the MaidSafe network, given the uni-directional nature of closeness meaning that even though A is one of B's closest neighbours, the reverse does not necessarily hold true. This highlights a limitation of theoretical analysis, in that it assumes nodes are able to identify the correct manager group for another node, despite not necessarily having knowledge of that area of the network.

Third, an area of centralisation has been identified in the MaidSafe design, allowing for the potential deliberate destruction or withholding of user data, namely around the chunk information holders. A group of chunk information holders was able to arbitrarily distribute data to a location of their choosing, potentially allowing for the selection of colluding nodes, rather than deterministic selection of nodes, which would help to prevent deliberate collusion. This would allow chunk information holders to initiate denial of service attacks on data they controlled. Another issue identified was the reliance upon the chunk information holders to approve mutation requests by reaching consensus on incoming requests — where a false consensus can be achieved, the data may be falsely mutated. It was also identified that chunk information holders could falsely request the removal of data from the network, by maliciously claiming it was no longer required, thus causing a loss of availability. As a result of this weakness, it was agreed with MaidSafe that the best mitigation was to prevent deletion of data until a suitable mitigation was identified.

Fourth, an attack was introduced, where a vault may utilise more storage capacity than they otherwise should, in order to prevent over-use of the shared storage resource on the network. In particular, a hand-off attack was identified, whereby a user can claim credit for storing large quantities of data, then subsequently re-transmit these to other nodes for storage, either in encrypted or obfuscated form. In the event of this data being requested, the rogue vault may request the retrieval of the copies it independently stored, and fulfil the request without needing to actually store the data, yet still gaining credit for storing the data on the network. To assist with this, a high performance proof-of-storage algorithm using a seekable stream cipher has been introduced, assisting the bootstrapping process by allowing third party verifies to efficiently ascertain that a given node is holding a set of arbitrary data, thus establishing credibility on the network.

Finally, a signature-backed transferable ownership scheme has been contributed, in order to allow users to re-key their accounts without losing access to addresses they already held their data at. The ability to re-key data is beneficial, in that it facilitates the routine rolling of keys as a good practice management strategy, and also facilitates the transfer of resources on the network between users. In particular, while other schemes for mutable data updating in decentralised networks exist, the ability to exchange an address between users has not been addressed, which would allow for the trading of desirable names.

Chapter 4 will now build upon the work of this chapter, investigating how untrusting parties can form contracts between themselves, and look at how an application would be built upon a decentralised storage network, and the feasibility of connecting to such an application from a mobile device.

Chapter 4

Decentralised Service Implementation & Performance

4.1 Introduction

One of the main advantages of centralised, commonly web-based, services, as discussed in Section 2.2, is that they offer convenient mobile access to a user's data, wherever they may be. In contrast, when using cellular-based data connections to access decentralised services, a number of new challenges are introduced, including considerations as to the overheads of maintaining and keeping alive connections between other members of the network, and handling incoming requests for data.

When considering decentralised services from the perspective of mobile users, it is important to consider the practical aspects of gaining access to storage or resources — without an always-online node being online on behalf of the user, they are unable to contribute resources to the network. Under a model such as MaidSafe, which is based on the premise of matching user demands against contribution, as discussed in Section 2.6.2, this poses a challenge for users. If a mobile-only user is unable to benefit from the security and privacy benefits of decentralised services, this will significantly restrict the reach of such services to those users. To avoid this limitation, it would be possible for one user to *offset* the storage requirements of another — there is no technical reason why the user must themselves provide the storage; merely that users do not take advantage of the generosity of others, as discussed in Section 2.6.1.

This chapter explores practical considerations as to application of decentralised services. In particular, it focuses on building a secure protocol for the creation of a digital contract, allowing one party to purchase storage from another party. This contract may be verified in the absence of a trusted third party, thus allowing a purchase of storage to take place. In the event that the provider of storage fails to offer storage as required, the buyer can be absolved of responsibility given the presence of such a contract, and the reputation system of the network may reduce the reputation of the storage provider as a result. A scheme to formulate the necessary operations within a decentralised network to provide user-facing services is also considered. Finally, in light of the performance issues identified when attempting to access decentralised services from mobile devices, a relay-based approach to accessing decentralised services is considered, to improve and make more practical access from mobile devices.

The contents of this chapter is based upon a number of published works [A3, A7, A8, A12]. Thanks and credit are also duly given to Pierre-Louis Dubouilh, who assisted with measuring performance of current decentralised DHT-based solutions as part of his Bachelors' degree project.

4.1.1 Chapter Contributions

This chapter presents the following 3 contributions.

- A decentralised digital contracts solution, allowing for third-party verification of contracts within a decentralised environment. This is demonstrated to allow users of a decentralised storage network to form purchase agreements for storage within the network, and others able to verify this agreement is honoured. A threat model of this protocol is presented, along with an extension allowing for bilateral obligations to be included in such a contract, rather than simply obligations upon the provider of the service.
- A scheme for the implementation of decentralised services is presented, identifying the necessary high-level operations, and showing how these should be implemented to realise secure, user-facing functionality for decentralised services.
- A hybrid relay-based approach is demonstrated as a viable solution to the problem of performance of mobile devices within a decentralised network, allowing mobile devices to access a decentralised service with the same relative ease as a regular centralised HTTP service, through the use of untrusted relays.

4.2 Storage Contracts

Fundamental to ensuring the flexibility of a storage-based network is the ability for a user to store files, even when they perhaps are unable to contribute towards the storage of files for other users. In the case of centralised storage systems, users typically pay the service operator for the allocation of a larger storage quota. In the case of a decentralised network however, this is not necessarily feasible, since data is split into chunks. These chunks are distributed to many vaults, with chunks redistributed throughout the network as nodes join and leave. As discussed by Ngan et al. , it is desirable to allow users to exchange storage quotas, such that a user with a surplus of storage can agree to sell that to a user who wishes to store more data than they can themselves store [75]. In a decentralised storage network, this matching of users means that sufficient storage will be present to ensure the longevity of the network, while offering users flexibility. This problem was discussed in more detail in Sections 2.6.1 and 2.6.2.

4.2.1 Motivation

Some users may have poorer internet connectivity than others, or they may be planning to travel for an extended period of time, therefore leaving their vault turned off. This gives rise to a desire for users to contract others to provide storage on their behalf. In a decentralised and trust-less network which requires all users to participate reciprocally, it is therefore necessary to identify a means through which a contract for storage could be enforced, such that the provider (rather than the buyer) would lose reputation for failing to provide the storage as agreed, and to ensure that neither party can lie about the terms of the agreed storage deal. A protocol, implemented on top of a decentralised storage network is therefore proposed, which would permit irrepudiable and non-forgeable agreements to offset storage to be negotiated, and enforced, by the decentralised network. This forms the basis of decentralised contracts, which will be verifiable by third parties, while facilitating privacy between parties negotiating the agreement.

4.2.2 Decentralised Storage Purchase Protocol

The protocol operates as follows:

- Buyer locates a provider offering storage for sale (this could be provided as a decentralised service listing buy/sell offers)
- Buyer and provider negotiate a price for an agreed quantity of storage, duration, price, and offer validity
- Provider signs this offer of sale with their private key, and states a (for example) Bitcoin address in the offer
- Buyer reviews the offer, ensuring the offer is as agreed

- To complete the deal, the buyer pays the agreed funds to the provider via the agreed Bitcoin address
- The sale is verifiable by any third party able to view the contract, since the Bitcoin transaction is visible on the blockchain

The sale agreement can be made available at any point in the future by the buyer, allowing other nodes to verify that it has completed a transaction with the provider of the storage. Since there is no requirement for the disclosure of the agreement prior to this point, there is no requirement for the transaction details to be made public unless there is a dispute. The provider would be responsible for storage provision, and their own reputation would be affected in the event of a shortfall affecting this user, as the user could present this contract to indicate the provider had agreed to contribute storage on their behalf. While Bitcoin is used above as an example for payment processing, this could easily be adapted to any cryptographic currency permitting third-party verification of payments via a public ledger (like the Bitcoin blockchain), and some method of timestamping (like the block ID).

The provider is protected from a malicious buyer, since the buyer cannot forge the seller's signature. This protects the details of the offer. It also states the identity of the buyer (their public key), meaning that another user cannot attempt to claim their entitlement to storage from another user's agreement. The offer contains a validity period (to prevent a buyer from requesting an offer, waiting a significant period of time until the value of storage or currency has changed, then accepting it), by specifying the transaction must take place before a given block ID on the blockchain.

The buyer is protected from a malicious provider, as the provider cannot deny the agreement exists (as the agreement was signed by the provider), and anyone can verify the agreed fee was paid, before the offer expired, thus validating the contract and its content. Note however that it is not possible for the seller to prove the existence of a contract with a given buyer; the buyer will always have deniability, since a seller could generate any contract they desired, and sign a copy containing any arbitrary terms.

As such, this protocol allows two parties, neither of whom need trust each other, to negotiate a deal for one to provide storage on behalf of the other party. Both parties are protected from the other failing to honour their part of the deal. In the event of the provider reneging on the deal, they can be held accountable by the decentralised network's reputation structure, as the agreement is provable. The MaidSafe network implementation of decentralised reputation would therefore handle scenarios where a provider reneges on a deal.

It is important to note that this network remains decentralised — purchasing storage from a user does not alter or influence where that data is stored — it is simply an offsetting process to ensure further storage exists in the network, to avoid storage space being depleted. Buyers are protected from price-fixing as they can purchase storage from any party on the network, without any inconvenience of moving their data around. As any user can already provide any quantity of storage to the network, this protocol does not reduce decentralisation, rather it increases flexibility for users, by allowing them to "shop around", thus encouraging a free market for pricing or terms.

4.2.3 Threat Modelling of Protocol

This proposal can be threat modelled by considering the relevant actors, and their potential actions within the protocol. The three actors are the seller, the buyer (or potential buyer, while negotiations are taking place), and any third party. During the process of negotiation, a potential buyer and potential seller may present each other with offers of storage capacity, for a duration of time, for a given price. These may or may not be of the form of signed offers; the negotiation itself does not form a contract, and if signed offers were to be used, it would be possible for the potential buyer to invoke the contract by making payment as specified within the offer. At this point, the buyer could present any of the previous signed offers — for this reason, the seller should use a different payment address in each offer, to prevent substitution of contracts. Even if substitution was to take place, the buyer would only be able to present contracts from earlier in negotiations, which they may have accepted at the time; since they could have accepted that offer at the time, and offers have an inbuilt expiry time, as discussed above).

A malicious or dishonest buyer (or potential buyer) cannot modify the content of a contract, as to do so requires the production of a valid signature, signed by the seller (storage provider). A dishonest seller cannot repudiate a contract, since the contract offer is signed using their private key, indicating that they or someone with access to their private key produced the offer. Since every participant in the Distributed Hash Table (DHT) network is identified by their public key, it is possible to retrieve the provider's public key for the purpose of signature verification.

In the event that a seller attempts to deny a contract ever existed, the buyer merely needs to reveal the contract agreed, and any third party may verify three factors — firstly, the public key of the seller from the contract statement, secondly whether or not payment was made (by examining the receiving address for transactions made to it of the required amount), and finally the terms agreed in the contract (i.e. quantity of storage agreed and the duration it would be provided for). Since whether or not payment was made is able to be ascertained by any third party from the blockchain, and the origin address of the payment confirms the party disbursing the funds, it is possible for any third party to validate the validity of the buyer's claims.

Privacy is preserved from all third parties, since there is no requirement for the existence of the contract to be disclosed at any point in the process, unless either of the two parties of the transaction makes available the contract. While an arbitrary contract may be generated and presented by the seller at any time, with retroespectively selected terms or details, even where a contract did not exist, this does not affect the process, since payment would be concluded at the point of acceptance, and therefore the contract is to protect the buyer.

To consider the scenario offering each party maximum gain, the optimal scenario for a buyer to deceive a verifier is to present either:

- a contract they have not paid the appropriate fee to activate
- a contract which was between another user and the seller, to which the buyer is not a party
- a contract with arbitrary terms designed to favour the buyer.

The first scenario is avoided by the need for payment to be made, by the agreed time (based on block number), to the given address. This payment, and its value, may be verified by the blockchain by any third party, preventing falsification of the payment existing. The second scenario is avoided by any third party verifying the contract to require a signed attestation from the buyer, which can be verified. This can be carried out in one of two ways — the buyer could be asked to sign an arbitrary message featuring an arbitrarily and randomly selected nonce by the verifier, using the private key of the Bitcoin (or other cryptographic currency) address that made the purchase, or the contract specification could require that the contract state the decentralised network address of the buyer, which they would be able to sign a message with.

The scenarios offering gain for a deceptive seller would be:

- to deny the existence of a contract, and claim the payment was unrelated, or a gift
- to claim that funds were not received, and back out of the contract, secretly retaining the payment
- to present a differing set of terms to those offered to the buyer, and insist these less favourable terms were the ones agreed to.

The first scenario is avoided, since the onus is only ever with the buyer to produce the contract. Since it is signed, the seller cannot believably deny its existence, or state the terms are falsified. The second scenario is avoided, as the payment is used as the act of signing the contract by the buyer, and thus accepting the offer. Provided the payment takes place using a method allowing for third party verification of receipt of payments, such as Bitcoin with its blockchain, it can be ascertained by any interested third party, having viewed the buyer's copy of the contract, that payment was indeed made per the agreement. The final scenario is avoided because this contract protocol only facilitates the placing of obligations upon the seller; the buyer's sole obligation is to provide payment if they wish to accept the agreement. In the event of contradictory versions of contract being presented, the one most favourable to the buyer (which any rational buyer would present) would be accepted by the third party verifier. A buyer presenting a less-favourable version of the contract would only be detrimenting themselves.

4.2.4 Adding Bi-Directional Obligations to Contracts

Note that if bi-directional contracts are desired, such as where there would be obligations upon the buyer, this can be achieved through a small modification to the protocol. It is necessary to obtain the digital signature of the buyer on the contract, in order to hold terms against the buyer. It is assumed that a rational buyer would only sign a contract they had accepted — a buyer signing other contracts arbitrarily would be irrational, and could be held to these (provided some party made payment), in the same manner in which someone who signs any physical contract they are presented with could be held to the signed agreements. The buyer's signature on the contract allows either party to present the contract, and invoke third-party adjudication of the situation. Note that in this scenario, the process of offers and contracts should be more carefully considered. To prevent a buyer from signing, then accepting (by providing payment) the contract without providing a signed copy to the seller, therefore depriving them of the ability to prove the buyer's acceptance of the terms, the process of the offer must be clarified.

The process forms that of a four-way handshake — The buyer solicits an offer from the seller, and specifies their requirements. The seller returns a signed offer, indicating a price, stating the terms of the offer including its expiry. If the buyer wishes to accept the terms, they should sign a response to the contract. This response should be appended to the offer, and should be a signature across the full offer (including the seller's signature). At this point, both parties have agreed the terms of the contract in principle, although this contract is not yet binding, as no payment information is provided. The seller then appends a payment request, including the Bitcoin or other payment address, to the signed response, and signs the resulting message, which now incorporates the offer, response and payment request. This must be conveyed to the buyer, such that they may verify the signature on the message is consistent with the party providing the offer, and then make payment. The payment may be verified by a third party, and the signed message incorporating the payment request now contains all the necessary information to allow a third party to verify the contract.

With a bi-directional contract, the seller can present the final message, which incorporates the terms signed by the buyer, as well as the payment information, to allow a third party to verify the contract was agreed. While a seller could potentially present the buyer-signed contract without the buyer having paid, this would only result in the buyer not having to pay, and being bound to a contract to which they had already agreed and signed, making this scenario no different than the uni-directional contract described previously in the threat model.

4.3 Service Implementation

The majority of today's services available to users are built around centralised architectures, as discussed previously in Section 2.1.1. These typically require users to divulge their information to the server, in order for the server to provide them access to their data, perhaps in more readable or usable forms, or simply through holding it so it may be retrieved from anywhere. This forms the basis of much of what is referred to by users as cloud storage, where the storage of their data is offloaded to a third party specialising in the storage of that data.

Another advantage of such centralised services is that they frequently offer access to user data and other functionality through just a web browser, or a mobile application. These typically access the backend user data through an API, able to be queried by the application for specific requests.

One significant trade-off here, however, is that for data to be available to a user via any web browser, it is usually necessary for it to be held by the service provider in a manner which renders it readable to the service operator. While this data may be encrypted when stored by the provider, it would be readable to them, since they would hold the keys needed to return the data to a client's web browser session.

The two key risks identified in Sections 2.1.1 and 2.2 were those of unauthorised access to the data held by the service provider, or that where the service provider ceases to provide access to data, as a result of a failure, or decision to no longer offer the service.

An example of the former situation would be compromise of the service provider, or where the service is a false-front to gather user data. An example of where a service provider ceases to provide access to data would be that of Google Reader, where Google decided to cease providing the service, requiring users to identify another service to use before their data was removed.

Decentralised services can solve both of these problems — within the context of this work, they are inherently designed to encrypt user data such that only the user may access it, so that user data may be safely stored on untrusted systems. Additionally, given the decentralised nature, it is not possible for the failure or decision of a single entity to result in the user's inability to continue to access their data.

The aim of this section is therefore to show that it is possible to operate services, which feature functionality familiar to users, which are not reliant upon a single entity's security or ongoing operation for utility of the service to be maintained. This empowers users to retain access to, and control of, their data at all times, and prevents another single party from depriving them of this utility.

This chapter will focus on decentralised network services backed by a storage network, which holds key-value pair data. This allows for any DHT-based storage network to be used, since DHTs are inherently key-value oriented, and allows services to be implemented on platforms such as the MaidSafe network, or indeed the mainline Kademlia DHT.

4.4 Service API and Storage Model

In order to demonstrate service implementations under a decentralised network, the following primitives are proposed for use. Each primitive is designed to represent user-facing functionality, required in order to allow for high-level applications to be developed upon such a network. Each API call is given a verb, which shall be represented as `VERB` within this section, to highlight the relations between such verbs.

4.4.1 GET Verb

The `GET` verb is designed for the retrieval of data from the decentralised network, based upon its address. This verb should be executed with a SHA-512 hash supplied as the address from which to retrieve data. After retrieval, the data should be verified — the SHA-512 hash of the data should be the same as the address which was requested, since content is addressed by its hash within DHT-based networks.

Note that the `GET` verb implicitly assumes that the user possesses the necessary information in order to gain access to the chunk, and decrypt it as required. This is because a secure decentralised network stores all user data in encrypted form, since the content of chunks is exposed to other, potentially untrusted users of the network. The `GET` verb therefore decrypts data from the requested address, using the key held in the appropriate data map, which was queried in order to locate the desired data at the address specified. More detail of the implementation of the MaidSafe implementation of the data map is given in Section 2.5.1.

4.4.2 PUT Verb

The PUT operation is designed to allow a user to upload data to the network, either for storage or for sharing with another user, which is a special form of storage, where the key is shared with another user, as discussed below for the SHARE verb.

A PUT operation accepts a chunk of data, and stores it at the address defined by either an optional address supplied in the case of mutable data, or at the address of the chunk's SHA-512 hash, in the case of immutable data where no address is supplied. There is an implicit step whereby the application calling this API takes measures to ensure the key used to encrypt the data stored by a PUT operation is held and accessible by the user. Ordinarily this would be by storing the hashes of each block of the data, and the ciphertext, within a data map for the file, and storing this with a key held within the user's root data atlas, as discussed in more detail in Section 2.5.2.

4.4.3 UPDATE Verb

The UPDATE operation may be carried out at any time following the PUT operation against a given address, provided this address is holding mutable data. In the event that data is immutable, this operation will fail. Mutable data may be used for data which will be modified in future, such as a data atlas or data map, or in any other scenario where it is desirable for the underlying application to hold data at an address *owned* by a given user. Note that in order to UPDATE a given address, it is necessary to provide an update ciphertext, signed using the same private signing key as used to sign the original data. If an invalid signature is supplied, nodes on the network will refuse to accept the updated version of the data. A secure protocol for updating of data is given in further detail in Section 3.10, which also securely facilitates the update of owner for a given piece of data within the network, preventing roll-back attacks, and allowing for secure re-synchronisation by an offline node, without requiring this node to trust the current state of the network.

4.4.4 DELETE Verb

For mutable data, it may be possible to invoke a DELETE operation on a given address. Note that MaidSafe have previously indicated they do not intend to permit the deletion of data, as a result of some of the potential attacks detailed in Section 3.3, which were highlighted to raise the possibility of data deletion by unauthorised parties. Nonetheless, for the purpose of completeness, the DELETE operation is considered here. A deletion may be considered as a special case of the UPDATE operation, in that it must be signed by the same key used previously to hold the data, or in the case of an implementation allowing for update of ownership records, by the key most recently indicated as owner. In this scenario, a deletion of data would result in the previous owner publishing a new record to the key as an ownership update, granting control to a randomly-generated private key, with the new contents of the chunk containing this private key, thus permitting any other user to re-gain control of this chunk by using the supplied private key to sign a new ownership message passing control to a key only known to themselves. The `DELETE` operation therefore removes the previous content of the value although, as with all digital systems, it should not be assumed that all previous copies of the data have been erased.

4.4.5 SHARE Verb

The share verb is one proposed to facilitate the secure and confidential sharing of information between two users of the network. Any user who has knowledge of the public key of another user may share a message with them. Using this process, the sender encrypts the message using the public key of the recipient, and stores the information at a deterministic sharing address, derived from both of their public keys. Note that, for the case of a user sharing with another user, whom is not aware of the sender's public key (such as an unsolicited message), it is possible for the message to be delivered to an append-only data record, held at an address derived from the recipient's public key.

For the scenario of user A sharing data with user B, user A would `PUT` the data, encrypted with user B's public key, as mutable data at the network address $H(A|B)$. In the event that user A wished to send more data, before user B retrieved the existing data, user A may `GET` the existing value of $H(A|B)$, and `UPDATE` it with a further messages contained within the structure.

This allows for secure, decentralised, asynchronous messaging between two users on the network. Previous works focusing on service implementations across DHTs have typically viewed messaging as a real-time or synchronous process [169], although the ability to carry out asynchronous messaging has been highlighted as a desirable property within decentralised communications architectures by the IETF [170]. Previous work on implementing asynchronous messaging across DHTs has identified a need for this, and an implementation was presented, although it used a single notification channel for a recipient, potentially raising challenges over how a user may receive multiple messages from multiple senders, over a period of time [171]. In contrast, this approach resolves these limitations, by creating an inbox per sender-recipient pair, and having a separate inbox for unsolicited contact. This approach also facilitates asynchronous acknowledgement of receipt of a message, so an offline sender may still become aware the recipient has come online and received their message.

When user B comes online, one of their first tasks will be to check for any incoming messages on these pre-arranged addresses. Therefore user B will carry out a `GET` query against the address $H(A|B)$ to determine if any messages have been sent by user A. If a message is present, it will be retrieved and decrypted using user B's private key.

To provide confirmation to user A that the message has been retrieved, it is possible for user B to provide user A with an acknowledgement. This is not mandatory, and will reveal that user B has been online since the message was sent, so there may be scenarios where a user may not wish to reveal this information. To acknowledge the message, user B carries out a `PUT` or `UPDATE`, depending on if a previous acknowledgement has been sent, to address $H(H(A|B))$. This is the hash of the inbox address queried by user B. As this is deterministic, user A can therefore also determine this address, and therefore may retrieve the acknowledgement, signed by user B.

4.4.6 GETSHARES Verb

The `GETSHARES` verb is a helper, used to combine the computation of $H(sender|self)$, for a given sender who the user wishes to accept asynchronous messages from, and a `GET` operation against the same address. This instruction may be used by user B, with the parameter of user A's address, in order to identify any messages shared by user A. `GETSHARES` may also be used to send an acknowledgement to $H(H(sender|self))$, as described for `SHARE`.

4.4.7 GETADDRESS Verb

The `GETADDRESS` verb is provided as a helper to locate a DHT address hash based upon a human-readable friendly-name, as proposed originally by MaidSafe [164] as selectable identities. Within the context of this work, such a selectable identity would be stored as mutable data. For a given parameter friendly name, N, $H(N)$ is calculated, and a `GET` operation is carried out implicitly on $H(N)$, resulting in the retrieval of the current mapping of the friendly name to another DHT address. This allows for the transfer or re-keying of a given friendly name towards a new account. Therefore, if a user has told another that their username on the decentralised network is, for example, “Steve1”, another user may use `GETADDRESS(Steve1)` to obtain the current public key address which “Steve1” is using on the network. This key is protected against unauthorised modification by third parties using the signed, decentralised, transferable data technique, described in Section 3.10.

To register a friendly-name N on the network, a `PUT` request is used to store a mutable chunk, stored at the address $H(N)$, containing a signed statement by the current owner of the name, pointing towards the current user. Note that the key used to sign the message need not be the key owner declared in the message; the key used to sign the message is simply the key required in order to modify the record and change the destination of the friendly name. Techniques such as this, within decentralised networks, are inherently vulnerable to flooding or Sybil-like attack, whereby users may create large numbers of identities at trivial cost to themselves, thus occupying the desirable friendly-names, requiring that users select more complex friendly-names to avoid ones already occupied [76].

4.5 Performance of Decentralised Services

A fundamental factor in whether or not decentralised services will gain user adoption is their usability, as well as their performance. Previous works have investigated characteristics of web-based applications which affect usability, and highlighted, in addition to the expected factors regarding user interface and accessibility, [172], that latency affects the usability of services. Therefore, it is important to consider the performance (and thus latency) of services; within decentralised networks, previous study has highlighted the challenges of performance within DHTs, specifically around that of typically high latencies, and comparatively low throughputs [173].

Decentralised services are a long-existing concept, with research proposing decentralised services and strategies as early as in 1993 [174]. Many popular services, including Skype [175] and Spotify [176, 177], were originally built upon decentralised, peer-to-peer technologies. Both have since moved from decentralised architectures to centralised architectures [178, 179].

This raises a number of privacy concerns for users — while it is not possible to ascertain for definite the reasons as to these decisions, the decision by Skype to move away from a peer-to-peer architecture, towards a centralised one controlled by Microsoft, has faced criticism [180]. Indeed, it has since been shown that, following the acquisition of Skype by Microsoft, although not necessarily as a direct result of moving away from a peer-to-peer architecture, Microsoft now has the capability to read and filter communications over Skype, and that this is carried out on a regular basis to all users [181], and this has been shown to apply even to apparently encrypted communications [180]. In a truly decentralised system, designed to resist the tampering or influence of a single party, this would not be possible, since no one entity would have absolute control over the client software, meaning the user community could continue to use the peer-to-peer technique. This is one reason that it is important to ensure decentralised service implementations feature open source client code, such that users may maintain them, in the event of the corporate failure of their original developer, or a deviation from practices which users find acceptable.

The principal architect of Skype has however claimed that the move away from peer-to-peer technology within Skype was technical, and related to offering better connectivity to mobile devices, for which the previous peer-to-peer architecture was not ideal, particularly around use of mobile clients for asynchronous messaging [182]. This highlights that there are ongoing and current technical and practical issues relating to the use of decentralised, peer-to-peer technology for mobile access. Therefore, this section focuses on the performance implications of achieving decentralised service implementation. It shall first explore performance of some standard DHT-based protocols, and shall then consider an improved hybrid approach, designed to offer the best compromise between centralised and decentralised services.

4.5.1 Performance of Current Decentralised DHT-Based Services

To investigate the performance of decentralised services within a variety of real-world network configurations, network latency was profiled between a number of different servers, each located in a different region. Latency was considered specifically, as a result of the Kademlia routing technique being heavily reliant upon sequentially querying nodes in different parts of the network in order to discover nodes and data. This makes the protocol very latency-dependent, since queries cannot be completed until this has occured, and if destination nodes are geographically distant, the latency will also impact on the data transfer itself. Unlike in other networks, where nodes may be organised by their geographical proximity, DHT-based networks organise nodes by their addresses, meaning that nodes may be required to communicate across high-latency links routinely. This work comparing network latency across various servers highlighted a significant consideration for decentralised networks, where users with close addresses may be geographically distant — even on a 1 Gigabit per second dedicated network connection within a data-centre, round-trip latencies were found to vary significantly based on geographical locations. For example, the mean round trip latency between DigitalOcean servers in London and Amsterdam was as low as 8.24 milliseconds, while the mean round-trip latency from New York to Los Angeles was 86.8 milliseconds. A more extreme example was seen between London and Singapore, with a mean round-trip latency of 336 milliseconds. This is of particular significance within a decentralised network, where the process of retrieval of a particular value requires an iterative look-up process, through several nodes, some of which will undoubtedly be geographically distant in a diverse network of geographically distributed users.

The performance of decentralised services on mobile devices is also a significant consideration for the future — mobile devices are now more widely used for internet access than regular desktop computers, within the United States [183]. In the developing world, mobile phones are increasingly becoming the primary means of connectivity for a growing number of new users, with 1.9 billion high-speed smartphone internet subscriptions worldwide in 2014, and 6.7 billion total worldwide mobile phone subscriptions [184]. Previous work has explored the challenge of mobile access to DHT-based networks [185, 186], and in particular considered the power consumption implications [187]. One further consideration here, along with the finding that mobile devices cannot effectively act as full members of a DHT for more than around 2 to 3 hours due to power usage, is that this previous study was carried out on a WiFi network. Indeed, using the cellular network would result in even higher power consumption, particularly in a scenario where many small messages were being sent over a period of time, due to the cellular `suspend_backoff` timer, used to hold the application processor (AP) awake during periods of several sleep and wake cycles [188].

Due to the significant variations in power consumption, particularly over short-term processes, on modern smartphones carrying out multiple operations in the background, it was found inappropriate to attempt to profile the power usage of a device while participating in a decentralised network. Therefore, overall time taken to carry out a transaction with the network was considered, since this will define how long the radio or network interface remains active for, to carry out a given task.

An initial test was carried out to ascertain the performance of a service implemented upon a standard DHT. The two key parameters identified here were query latency and throughput. Query latency was the time taken for a `GET` request on the network, to retrieve a given value from the DHT, to be carried out. In the tests carried out, it was found to take approximately 12 seconds for the key to be retrieved from the network. This appeared in line with previous research, which has shown that DHT lookup performance is typically slow, since lookup performance is $O(log(N))$ for a network of size N [189]. As the mainline BitTorrent DHT was found to have up to 27 million users per day in research carried out in 2013 [53], this would indicate around 8 look-up cycles may be required to locate a given key on a large DHT. Throughput in transferring data across the DHT was found to vary directly with latency — with a latency-unconstrained ethernet connection having a 3.5 millisecond latency to the first external internet router, a throughput of 420 kilobytes per second was achieved across the DHT. By adding an artificial 100 millisecond latency to this system using the iptables firewall, the throughput fell to 50 kilobytes per second, on account of the large number of small packets being sent to locate the necessary nodes, and then to relay the outgoing data to these nodes.

Therefore, from the perspective of a mobile device, having to carry out multiple recursive lookups poses a performance challenge, as well as a practical challenge from a power usage perspective of cellular radio utilisation. The relatively slow throughput was also a significant consideration, as it will require the radio to be kept active for longer, in order to complete transmission of data.

To establish the performance within a constrained environment, for the purpose of experimental reproducibility, a simulation of a decentralised DHT-like network was created. Firstly, a client and server application were configured to communicate using UDP, in line with how state-of-the-art DHT implementations work [190] — the server was designed to simulate the DHT, without considering the routing latency presented by the DHT, such that the transport efficiency may be investigated.

A 1 MB file of random data from `/dev/urandom` was generated and selected, to be transmitted to the simulated network using UDP. A maximum UDP packet size of 8100 bytes was selected, since this was able to pass through the Linux networkng stack without being dropped. Note that while there is no definitive maximum UDP packet size, others have suggested the use of smaller UDP packets [1], which results in the requirement to transmit more packets, and therefore results in a greater time until transmission is completed. Therefore, these results present a best-case scenario for the transmission of UDP packets, for comparison, and it should be noted that performance gains achieved beyond this would be more significant when smaller packets were used.

Performance of the transfer of a 1 MB file was found to vary linearly with the latency introduced to the connection through iptables, with the transfer taking 1.15 seconds at no added latency, and as much as 78.6 seconds with 200 ms of added latency. This is shown in Figure 4.1.

Note that these results assume a simple, non-parallel process of retrieval is carried out, where each chunk was located in a different location, and retrieved sequentially. If an implementation featuring multiple network connection threads was created and used, performance would be improved, as the impact of latency on the overall performance would be reduced.

Clearly this presents a significant challenge, both from the perspective of service performance, as well as from the perspective of power consumption. The simplest means to mitigate it, based on the performance considerations from above, would be to ensure that the process of transferring the same data took less time, therefore requiring less transmission time over the cellular or WiFi interface.

[1] `http://www-01.ibm.com/support/knowledgecenter/SSB23S_1.1.0.10/com.ibm.ztpf-ztpfdf.doc_put.10/gtpc1/gtpc1mst45.html`

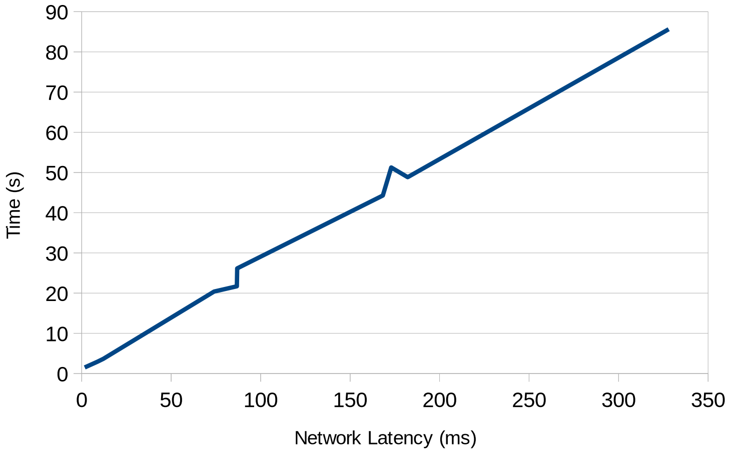


Figure 4.1
UDP performance with varied latency to send a 1 MB file

4.5.2 A Hybrid Approach to Accessing Decentralised Services

The most obvious approach to solving this challenge is to use a transmission protocol such as TCP, which offers features such as windowing, reliable transport, and fragmentation. These features typically offer significantly improved performance, albeit with the overhead of establishing a TCP connection, which is the reason they are not typically used within DHTs, and current DHTs used UDP as discussed above. There are heightened challenges in tunnelling through NAT when using TCP [191], meaning that a server with fixed public IP address would be needed to act as the TCP relay. This relay server would communicate with the remainder of the DHT over UDP as normal, in order to preserve the decentralised nature of the network. Therefore, users are dependent upon the relay, although may change relays as desired, since the process of relaying their requests to the decentralised network can be fulfilled by any relay.

In addition, by reducing or removing the delays required in order to access the decentralised network through the $O(log(n))$ lookup process, there would be benefits for mobile devices. Eliminating the need to hold open active connections with other peers would allow a mobile device's cellular radio or WiFi interface to sleep for longer periods of time, saving power on the device significantly over being a direct member of the network.

To achieve these savings, without removing the benefits of decentralisation to users, a hybrid approach to access decentralised services is proposed. Within this hybrid approach, the concept of an relatively untrusted relay server is introduced, in order to allow for a client to gain access to resources and data on the decentralised network, without entrusting any other party with their keys or network identity credentials. This allows a client application running on a device to access the decentralised network, as though it were any other regular web-based API, facilitating the re-use of the many existing and well-established libraries for connection to web-based API services.

Nonetheless, when adding any kind of fixed server to a decentralised network design, it is important to consider the implications of the creation of such a server, specifically around whether it would introduce a central point of failure for users, or whether the operator of the server may gain any kind of advantage on the network, specifically over allowing access to unencrypted user data, or falsifying requests on behalf of the user.

This proposed solution avoids the risks of centralisation within the network, by ensuring that these relatively untrusted relay servers are, as the name suggests, considered potentially malicious by the client software running on the user's device. The client software should not assume that the relay acts correctly, and should be able to detect any kind of interference, and switch to an alternative relay. To avoid centralisation, relays are federated, allowing a user to, in principle, connect to any relay server of their choice. Relays can by run by any party, since there is no inherent requirement for users to trust the operator. Nonetheless, to ensure continuity of access to services, it is important that they are run by a diverse range of users. One approach to this is to encourage users to self-host relay servers, an approach which has seen success in open source projects [2].

Here the motivations of a user for running a relay server shall not be considered, as many other systems have shown that people are willing to host servers out of goodwill to other users. For example, the Syncthing community provide a variety of relay servers for free to users, and allow anyone to host their own relay, which assists users behind NAT in using their synchronisation tool. Likewise, the Tor anonymity protocol allows anyone to host a relay or bridge, and indeed many companies choose to support the network by hosting such nodes [3]. Indeed, to preserve diversity of those offering relays, for resiliency purposes, there are benefits in encouraging users to run relay servers out of goodwill, rather than as a large-scale commercial service with single entity causing wide failure, although nothing here precludes this from happening, such as using business models seen by VPN providers, where free trials are available, and a small fee gives higher speeds and wider choice of relays which are available only to subscribers.

4.5.2.1 The Hybrid Service Connection Model

Within the hybrid service connection model, it is assumed that a client application is shipped with a bootstrap list of candidate relay nodes, or makes contact with the decentralised network at least once, in order to obtain a list of relay nodes. This ensures that the application may gain access to the decentralised network, either using a relay

[2] https://github.com/syncthing/relaysrv

[3] https://metrics.torproject.org/networksize.html

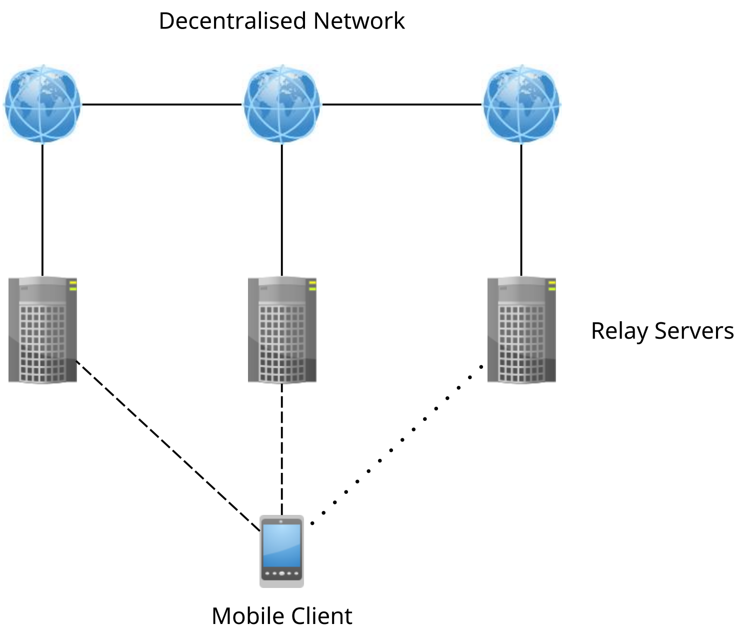


Figure 4.2
Architecture of relay-based mobile access

from a list of bootstrap relays, or by having the ability to join the decentralised network through bootstrapping to find a public list of relays. The architecture of this is shown in Figure 4.2.

Once a connection is established, it is possible for a client to carry out all network operations through a relay node. Here, the proposal is security-modelled to ensure that relays do not hold a privileged position which may reveal user data. All requests received by the network which would mutate or store data must be signed, per the protocol described in Section 3.10. A client shall make requests to the network by sending a request to the relay, which then relays the message to the decentralised network. This inherently places the relay node in a "man-in-the-middle" capable position, which requires consideration to be given as to constraining the relay's ability to adversely affect a client's communications.

Firstly, all data stored on the network is encrypted, thus ensuring that the relay cannot gain access to the data contained within chunks being relayed. This prevents the relay node from gaining access to the content of a user's data. When a client application requires data be uploaded to the network, the node holding the data will issue a signed response, acknowledging receipt of the data, incorporating the cryptographic hash of its content (which is the address for immutable data), and the identity of the node managing the storage of the data. This is to ensure that a relay has genuinely passed the request on to the network, and that the data held by the network was not tampered with or modified by the node.

One limitation of the use of a relay is that, within a decentralised network, the process of verifying the signature of the node holding the data is difficult — it is not possible to immediately identify the node holding the data, and verify that this is correct. A determined node could generate private keys corresponding to a large range of public keys, and use this to generate signatures from addresses appearing to be close to the address of the data, in an attack similar to that demonstrated in Section 3.3.2. Fundamentally, this attack is not easily mitigated without the client holding a large database of the current network's configuration, which in itself is not feasible due to the requirement to effectively hold an entire map of the network, which would not scale to a larger network; the fundamental goal of DHT routing was to create a scalable solution with logarithmic complexity, rather than one with linear or poorer routing complexity [52].

Therefore, this issue is not unique when a relay is introduced; a party determined enough to create false identities on a decentralised network can typically carry out a so-called Sybil attack [76] with relative ease, simply by generating new identities. As was shown in Section 3.3.1.2, this is a challenge inherent to the MaidSafe network, where it is assumed that users may verify the validity of a node claiming responsibility to store a given piece of data.

In a hybrid solution, however, a relatively straightforward solution exists to this challenge; a second relay, assumed to be non-colluding, may be used to verify the presence of the data on the network. This requires an element of trust to be placed by the user in the relay, meaning this not an entirely untrusted relay implementation, but the ability to receive independent confirmation of the presence of data is advantageous for users unable to act as full members of the network for power reasons. Indeed, a similar technique would be advisable on the regular decentralised network, since the MaidSafe network implements localised opportunistic caching of content on the network, which would allow a managing node to cache content rather than truly distribute it to the network.

Fundamentally, however, this concern can be mitigated within a decentralised networks by allowing a node to verify their own content is located within the network through its retrieval; unreliable relays may be reported by users as unreliable, therefore harming their reputation by clients. Relays must sign responses they issue to clients, to allow the client to have assurance the relay is willing to use its reputation to confirm it carried out the action. This allows a client to verify, and prove to another user, that a given relay server produced a falsified attestation that information was stored on the network — the client may verify if the data is indeed held on the network, and draw their own conclusions. If such a report was stored on the network in a deterministically-located manner, it is possible for other parties to view the reputation records for a given relay. For example, by storing reputation reports at addresses corresponding to successive cryptographic hashes of a given relay's address, users may identify reputation reports concerning a given relay. For a relay of public key R, it will have an address of $H(R)$ on the network. The first report of the relay's reputation may be made at $H(H(R))$, and the second at $H(H(H(R)))$, once the first address has been used, and so on. This allows for users accessing the network to make reports as to the reliability or otherwise of relays. Indeed, a relay could also suffer reputational damage as a result of falsely claiming that a given address does not exist on the network; the relay will return a signed attestation that the content was unable to be found, and a relay found to be falsely hiding the existence of reputation reports about itself or other relays could be detected by other users, and thus lose its trust in the community.

The process for verifying reputation reports can be automated, since it would simply require the parsing of a regular relay response, and carrying out independent verification of whether or not a request was indeed carried out. The following logic should be used to correctly process a relay response, to prevent a malicious client from falsely claiming a relay to be acting improperly on the network.

Firstly, for a claim of data not being stored properly on the network;

- Ascertain the address the relay claimed to update, and verify if this data exists on the network
- Identify if data is mutable or immutable. Immutable data may be verified simply by requesting the data and ensuring that it exists on the network. Immutable data is stored at the address of its content's hash, therefore it is self-verifying
- If immutable, check the hash of the content which was found at the address; if it exists, the complaint is false or the relay has since stored the data it was accused of withholding.
- If immutable, establish the current version of the data held; if the version held is greater than or equal to the version from the relay receipt, the complaint is false or since irrelevant. Otherwise, the complaint is valid.

Fundamentally, the relay does not gain access to the plaintext of a user's data at any point, and it may carry out its duties without requiring such access. All requests from the user for storage are merely relayed (in signed form) by the originator of the request, which is the client application. The relay is therefore not in a position to gain access to the contents of data passing through it at any point, meaning it is not necessary for the relay to be inherently trustworthy. While a client cannot necessarily be sure any given relay is passing requests to the correct network (a determined malicious relay could relay requests to a second, forked decentralised network), but none of the properties of the confidentiality of user data would be defeated here; such an adversary could potentially erase user data in future to cause denial of service, but reasonable precautions being taken would prevent this, such as using a second relay to verify the actions of the first, to establish the trustworthiness of the first relay, and looking for reports of misbehaviour from a relay on the network, using an unrelated relay to seek such reports, or while accessing the network directly.

To mitigate some of the risks of malicious relays, users may operate their own, and share them with friends or associates who also operate such relays, creating a series of independent relay nodes allowing access to the network. Confidentiality of all information transferred is preserved in such scenarios.

4.5.3 Performance of Hybrid Relayed Network

An implementation of this hybrid decentralised network was created, and used to determine the possible throughput when transmitting data directly to the relay server over a TCP connection. 8100-byte chunks (as used previously to measure UDP performance) were able to be streamlined into a single HTTP request to the hybrid API.

One significant improvement of the proposed hybrid approach to access to a decentralised network is that users may select their relays based upon geographical location. This is in contrast to the actual data on the network itself, which is uniformly distributed across the network, based upon the distribution of users providing storage. Therefore, a client application selecting the relay with the lowest latency will realise the fastest transmission times to their initial relay. This relay may then carry out the intensive operations on the network on the client's behalf, waiting for values to be retrieved, and then return a response from the network to the user application. The relay may optionally cache data from the network, particularly immutable data, in order to offer more rapid retrieval of content for users.

4.6 Conclusions

The practical considerations of a decentralised storage network have been considered, with a view to creation of applications upon such a network. Firstly, the research question of how parties who do not trust each other has been addressed with the contribution of a decentralised digital contracts solution, allowing for agreements to be made and enforced within the decentralised network. This was presented within the context of a contract for the purchase of storage on the network. These contracts are able to be enforced by third parties on the network, potentially allowing for fully automated agreements to be in place, with programmatic validation and enforcement. Secondly, a high level scheme for implementation of decentralised services has been presented, showing the high level operations available for applications to implement to build their functionality. Finally, to address the research question of the practicality of connecting to a decentralised storage network from a mobile device, performance of accessing decentralised storage was investigated across both UDP and TCP protocols, and a hybrid relay-based approach was then contributed as a result of the identified limitations in performance over connections with non-negligible latency. The contributed approach allows mobile devices to connect to an untrusted relay server to gain access to a decentralised network, without having to stay active on the network and thus incurring battery usage and high levels of network traffic.

This makes accessing and using decentralised services more practical from a mobile device, but also highlights the importance of ensuring the security of client devices, particularly when highly portable platforms such as mobile devices are being used. Chapter 5 will therefore consider the security of endpoint devices used to access a decentralised storage network, exploring the security of the Android platform, as well as other implementations of encryption within Android applications.

Chapter 5

Endpoint Security

5.1 Introduction

A key challenge of achieving secure services and storage is the security of the endpoint used to gain access to the service — if a device with access to sensitive data is able to be compromised, the data the device may access can then be accessed through the vulnerable device. Endpoint security shall be considered in the context of both client devices, such as computers, smartphones and tablets, as well as to service-side equipment, deployed by the service provider.

This chapter shall consider endpoint security of mobile computing platforms, from the perspective of attempting to gain a trustworthy operating environment. A number of security weaknesses in the implementation of Google's components of the Android operating system are explored, highlighting the risks posed to over a billion users of such devices, in the event that their network connection is untrustworthy. In addition, the capabilities exposed to Google, and the extent to which data is transmitted to Google over inadequately verified channels are demonstrated, and it is shown that Google can remotely enable its device management features on a device, to then change the encryption and lockscreen passphrase on the device, thus overriding a user's security settings on the device.

The importance of correct use and implementation of cryptography is also shown, with an analysis carried out across a number of popular Android encryption apps, which highlight the importance of well-reviewed implementations of standard cryptography, rather than through the development of custom solutions. This highlights the importance to privacy and confidentiality of user data of providing encryption of files at a platform level, rather than at an application level. The prevalence of relatively widely used Android applications, marketed as providing file encryption, which were easily broken highlights the risks of relying on the application software alone for security.

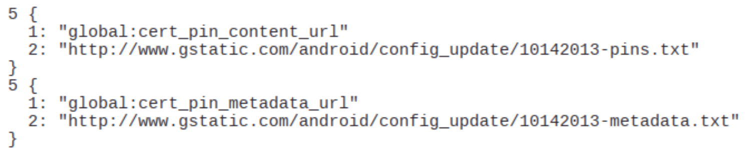
This chapter is based upon work published in [A4, A10, A14, A21], exploring the security of various client devices and software, identifying vulnerabilities, and ways to improve the security of client software and devices.

5.2 Security of Android Certificate Pinning and Device Provisioning

Android 4.2 introduced certificate pinning functionality as a security feature, as discussed in Section 2.10. As highlighted previously, Android 4.4 stated that this functionality was enabled for connections to Google's centralised mobile services platform, to protect users against connections being tampered with or spoofed by third parties with access to rogue CA certificates, as discussed in Section 2.10.3. This section shall detail work carried out to assess the operation of the Android certificate pinning process on a Google Experience device, to ascertain whether the implementation was sufficiently secure to protect users against the types of attacks which certificate pinning is intended to prevent. Since certificate pinning is used to protect the device setup and provisioning process (also referred to as check-in), the ability to compromise this, or other connections to Google's mobile services platform, may permit a third party attacker to exert control over client devices, which would violate the security of the client device.

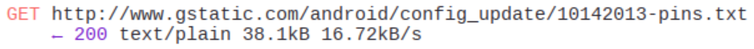
5.2.1 Client Provisioning Process

A newly configured Sony Xperia Z Ultra (Google Play Edition), running Android 4.4.4, was reset to the clean, pre-installed configuration of the operating system, with the only addition made being a single additional root trusted certificate authority. This was added to the internal operating system CA store, to emulate a scenario where a default trusted system CA has been compromised. It was then connected to the internet through a network configured to allow for interception and active tampering with the ongoing connection, as discussed in Section 2.10.1. Upon connection to the internet, it underwent an initial provisioning process triggered by Google Mobile Services. This involved the Android device establishing a connection to Google's provisioning servers over HTTPS and requesting provisioning data, while at the same time sending device information (including unique device serial numbers such as the device's IMEI number and MAC address). The received provisioning data configured the Android device, setting a number of system-only configuration options within a protected database on the device [136]. As part of this initial process, a number of URLs were delivered to, and stored on the device, for the purpose of future configuration updates. One of these configuration files pertained to a whitelist of digital certificates for various domain names. The purpose of this list is to increase user security by ensuring that connections to se-



```
5 {
  1: "global:cert_pin_content_url"
  2: "http://www.gstatic.com/android/config_update/10142013-pins.txt"
}
5 {
  1: "global:cert_pin_metadata_url"
  2: "http://www.gstatic.com/android/config_update/10142013-metadata.txt"
}
```

Figure 5.1
Android check-in data indicating plain HTTP updates



```
GET http://www.gstatic.com/android/config_update/10142013-pins.txt
    ← 200 text/plain 38.1kB 16.72kB/s
```

Figure 5.2
Plain HTTP GET request to retrieve certificate pinning data

cure websites are not being intercepted due to a Man in the Middle attack (MITM) attack, through the compromise of a CA. This list is referred to as the pin list within the operating system, as shown in Figure 5.1 This pinning update data was versioned and signed in a corresponding metadata file to prevent tampering, with the public signing key communicated to the client through the initial setup process.

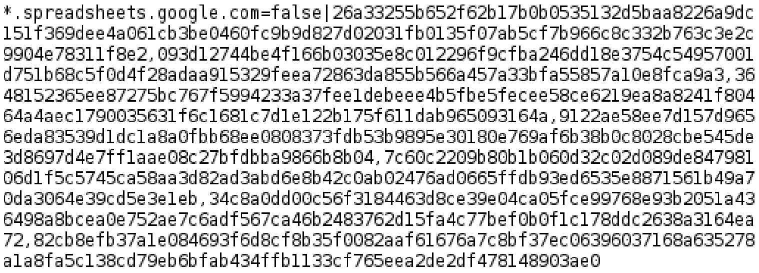
Note that, at this point, the interception of the check-in process was being carried out through an intercepted SSL connection, indicating that the check-in process was occurring without any form of certificate pinning validation. Therefore, any adversary with access or ability to issue a certificate for Google's check-in servers, signed by any valid CA as discussed in Section 2.10.2, could return their own responses and URLs for this part of the process, as well as intercept and monitor the entire device check-in process, using such a compromised connection. One example of an attack would be to replay outdated certificate pinning records returning disabled pins, such as those shown to exist here.

5.2.2 Certificate Pinning

As shown in Figure 5.1, the supplied pinning URL was a plain HTTP link, rather than a secure HTTPS link. The HTTP request observed through interception also indicated that this process was indeed being carried out over plain HTTP, as shown in Figure 5.2.

It was found to not be possible to tamper with the contents of the pin database, since it was signed and authenticated by the client, with a CA-signed public key used to validate this signature transmitted during the initial check-in.

The pinning data sent by the server was of the form shown in Figure 5.3 — each record contained a domain prefix, an enabled state, and a certificate pin record containing public key hashes. Of note was that the enabled state for every pin was `false`, therefore indicating that the certificate pins were not being enforced. According to the Android source code, a disabled pin is only used for logging, and is not used to block



```
*.spreadsheets.google.com=false|26a33255b652f62b17b0b0535132d5baa8226a9dc
151f369dee4a061cb3be0460fc9b9d827d02031fb0135f07ab5cf7b966c8c332b763c3e2c
9904e78311f8e2,093d12744be4f166b03035e8c012296f9cfba246dd18e3754c54957001
d751b68c5f0d4f28adaa915329feea72863da855b566a457a33bfa55857a10e8fca9a3,36
48152365ee87275bc767f5994233a37fee1debeee4b5fbe5fecee58ce6219ea8a8241f804
64a4aec1790035631f6c1681c7d1e122b175f611dab965093164a,9122ae58ee7d157d965
6eda83539d1dc1a8a0fbb68ee0808373fdb53b9895e30180e769af6b38b0c8028cbe545de
3d8697d4e7ff1aae08c27bfdbba9866b8b04,7c60c2209b80b1b060d32c02d089de847981
06d1f5c5745ca58aa3d82ad3abd6e8b42c0ab02476ad0665ffdb93ed6535e8871561b49a7
0da3064e39cd5e3e1eb,34c8a0dd00c56f3184463d8ce39e04ca05fce99768e93b2051a43
6498a8bcea0e752ae7c6adf567ca46b2483762d15fa4c77bef0b0f1c178ddc2638a3164ea
72,82cb8efb37a1e084693f6d8cf8b35f0082aaf61676a7c8bf37ec06396037168a635278
a1a8fa5c138cd79eb6bfab434ffb1133cf765eea2de2df478148903ae0
```

Figure 5.3
Pinning data for a single site, from the HTTP-served configuration file

connections where pinning fails [1]. This indicates that, since devices were receiving only disabled pins, user data would not be protected when pinning violations were detected.

Nonetheless, any adversary with the ability to intercept network traffic would be able to intercept this specific HTTP request, and filter it, either blocking the request, returning an error, or responding with an old database, to prevent a future update from taking place. Significantly, no certificate pins were shipped with the device — the `/data/misc/keychain/pins` file was not present on a clean device, since the `/data` partition is erased during a factory reset [2]. Since no certificate pins were shipped with the device, no protection from certificate pinning was available out of the box. Even with pins configured to notify Google, these notifications were not issued, since no pinning records were available on the device.

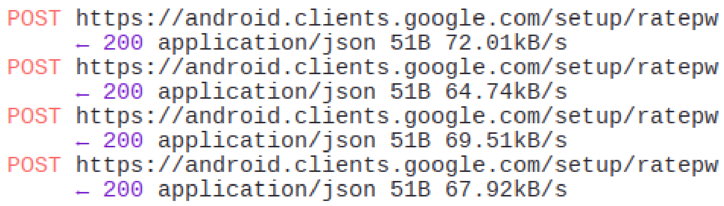
Since no pins were shipped on the device, any party with network-level access would be able to return a 404 error on attempts to update this pinning list. Indeed, by configuring mitmproxy to return a 404 error upon requests to the certificate pinning URL, there was no visible sign of failure, nor any alert to the user that their connection may be tampered with, or that no certificate pins had been loaded. Had certificate pins been updated over an HTTPS link, this would not have been as easy to carry out, and it would have required a valid certificate. This attack is therefore viable for network operators, or indeed countries with access to a national, and relatively centralised, internet gateway, even if they do not have access to SSL certificates for interception purposes, and it is of high impact, since it prevents Android devices from receiving any pinning data. As users are not notified of this failure, they may use their device normally, without being aware of the lack of protection. This may be used by a malicious entity to leverage incorrectly-issued SSL certificates which would have otherwise triggered CA pinning alerts.

Significantly, since the logging feature requires a pinning list to be present [3], such an

[1] http://androidxref.com/4.4.4_r1/xref/libcore/crypto/src/main/java/org/conscrypt/PinListEntry.java#40

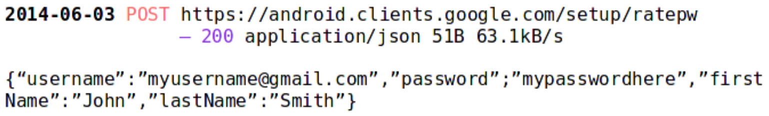
[2] https://android.googlesource.com/platform/bootable/recovery/+/kitkat-release/recovery.cpp#104

[3] http://androidxref.com/4.4.4_r1/xref/libcore/crypto/src/main/java/org/conscrypt/PinListEntry.java#40



```
POST https://android.clients.google.com/setup/ratepw
      ← 200 application/json 51B 72.01kB/s
POST https://android.clients.google.com/setup/ratepw
      ← 200 application/json 51B 64.74kB/s
POST https://android.clients.google.com/setup/ratepw
      ← 200 application/json 51B 69.51kB/s
POST https://android.clients.google.com/setup/ratepw
      ← 200 application/json 51B 67.92kB/s
```

Figure 5.4
Android setup process sending password rating traffic to Google servers



```
2014-06-03 POST https://android.clients.google.com/setup/ratepw
                ← 200 application/json 51B 63.1kB/s

{"username":"myusername@gmail.com","password";"mypasswordhere","first
Name":"John","lastName":"Smith"}
```

Figure 5.5
Android setup process sending current password being typed to Google servers

attack would not be detectable by Google through the pinning reports, since devices would not have any pins loaded, and therefore would not report any failures.

5.2.3 Account Registration Security

During the process of setting up an Android device, users are encouraged to either create a new Google account, or log into an existing one. It was observed through interception of this traffic that during the process of registering a new account, the registration process sent a message upon each change of the password field to Google's servers, as shown in Figure 5.4. While this was sent over HTTPS, the fact this data could be captured using a system-installed CA indicated that an attacker with access to a rogue CA would also be able to capture this. These recurring requests appear to indicate that the traffic is being used for the purpose of rating the user's password.

By examining these messages, it became clear that the setup process was transmitting to Google the plaintext content of the text box, upon every change to its value, as shown in Figure 5.5. Indeed, this request was found to contain the selected Gmail address of the account being created, the current content of the password box, and the first and last names of the user, as supplied to the setup process.

Despite this, there would appear to be no clear reason for the chosen Gmail address, or indeed the user's full name, to be transmitted as part of the process for measuring password strength. Indeed, doing so made it easier to filter only traffic containing plaintext passwords, and to also receive the user's name and chosen email address, which would potentially be useful to a malicious party attempting to gain access to users' email accounts. There also appears to be no clear justification for carrying out this process online — offline password strength meters are widely available and implemented for various platforms [4], removing the need to send passwords in plaintext.

In light of the goal of a password strength meter being to encourage users to use a stronger password, after they perhaps entered a weaker one, this appears to be a badly designed system, which may result in users inadvertently revealing other, weaker passwords, which they may commonly use. With 55% of adult internet users in the UK admitting that they "use the same password for most, if not all, websites" [24], revealing the plaintext contents of this password entry box, prior to the user selecting a password, poses a significant risk to users, if their connection was being intercepted, as previously discussed. Since everything typed into the box is sent to a remote server, this poses a significant risk that more than one password as used by an individual may be compromised, in the event their traffic was undergoing a man-in-the-middle attack.

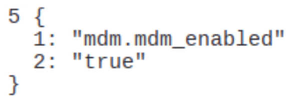
Additionally, on two other occasions within the registration process, the chosen user password was sent to Google over HTTPS, in a fully readable form. Given the lack of certificate pinning, this puts users at significant risk during the device setup process, as they enter passwords. Techniques avoiding the need for the trust of the server when handling passwords have been introduced in previous works [192], and these could assist in eliminating the transmission of plaintext passwords over a potentially insecure network. For example, client-side hashed passwords could be sent to the server, or Secure Remote Password (SRP) verification data could be sent by the client to the server during registration.

5.2.4 Remote Control of Device

During the check-in process, Android was found to obtain configuration information from Google's servers, as discussed in Section 5.2.1. One of these configuration parameters pertains to the status of the "Android Device Manager" feature, which is a part of Google's system application suite, that does not form part of the open source core of Android. This device manager service offers the ability to locate and remotely lock or wipe a lost or stolen Android device, by using the associated Google account. Android Device Manager is a piece of software commonly used for Mobile Device Management (MDM).

From analysis of the data exchanged during the check-in process, it was determined the Android Device Manager tool was able to be remotely enabled. On a freshly-reset Sony Xperia Z Ultra, it was first verified, while offline, that Android Device Manager was disabled in the Security settings menu by default. After connecting the device to the internet, it was noted (without having logging into a Google account or similar) that the check-in process had silently enabled device administrator privileges for Android Device

[4] https://github.com/dropbox/zxcvbn-ios



```
5 {
  1: "mdm.mdm_enabled"
  2: "true"
}
```

Figure 5.6
Google check-in process remotely enabling Android Device Manager

Manager. The received request for enabling device manager is shown in Figure 5.6.

As it was therefore possible for Google to use the check-in process to remotely enable the device administrator features of Android Device Manager, and this process completed before any login or account creation was carried out, the Android Device Manager feature may pose risks to users during (and after) the setup process. By default, Android Device Manager allows devices to be located and remotely locked or wiped. The process of remote locking, however, was identified to be of potential interest for this security analysis, as it exposed a major limitation of the Android security model's reliance upon Google as a trusted third party. Namely, if a user loses their device, they can log into their Google account online and carry out a variety of operations, including setting a lockscreen password to protect their device in case it is found or in the possession of a malicious party. In the event that the device already has such a password, it is overridden and replaced with the new one.

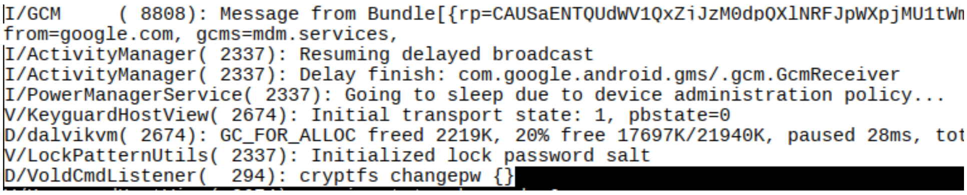
Android devices also feature device encryption functionality, designed to protect data from attack in the event of the theft of a device, or access by an unauthorised party. Android data encryption would require the device password to be entered at power-on, in order to decrypt the data held on the device. The password entered here is the same as that entered on the regular device lockscreen, and therefore the two are linked, as is clear from a long-running feature request to allow the separation of the two features [193].

On account of this, a hypothesis was formulated, that the Android Device Manager functionality would most probably make use of the regular Device Administrator APIs permitting the changing of the user password. This theory was formed on the basis of the regular password change functionality triggering an update of the encryption password [5].

To test this hypothesis, a test was constructed, whereby the Sony Xperia Z Ultra was encrypted using a password, and logged into a Google account. While monitoring the system debug logs, a request to remotely lock the device was issued through Android Device Manager's web interface. Figure 5.7 shows the device logs gathered during this process, indicating that the process also changed the device encryption password, as a result of a remotely-initiated message from Google's servers.

The device was then rebooted, and the original password attempted, as well as that

[5] http://androidxref.com/4.4.4_r1/xref/frameworks/base/core/java/com/android/internal/widget/LockPatternUtils.java#622



```
I/GCM      ( 8808): Message from Bundle[{rp=CAUSaENTQUdWV1QxZiJzM0dpQXlNRFJpWXpjMU1tWm
from=google.com, gcms=mdm.services,
I/ActivityManager( 2337): Resuming delayed broadcast
I/ActivityManager( 2337): Delay finish: com.google.android.gms/.gcm.GcmReceiver
I/PowerManagerService( 2337): Going to sleep due to device administration policy...
V/KeyguardHostView( 2674): Initial transport state: 1, pbstate=0
D/dalvikvm( 2674): GC_FOR_ALLOC freed 2219K, 20% free 17697K/21940K, paused 28ms, tot
V/LockPatternUtils( 2337): Initialized lock password salt
D/VoldCmdListener(  294): cryptfs changepw {}[redacted]
```

Figure 5.7
Android Device Manager changing device encryption password remotely

of the new password. The original password was no longer capable of decrypting the device, and only the new password, sent over the internet from Google's Android Device Manager servers, was able to be used. This raises two concerns — firstly that a malicious party could compel or social engineer Google to change the password on a device remotely, and secondly that a malicious party gaining access to Google's systems could carry out a denial-of-service attack on users, thus locking them out of their phones by changing their password.

5.2.5 Summary of Findings

A number of insecurities in smartphones and tablets running on the Google Android platform have been highlighted. It has firstly showed that the implementation of certificate pinning without using an HTTPS connection can be easily blocked or tampered with, thus preventing a device from loading certificate pinning data. Without any certificate pins loaded, there is no constraint on which CAs may present valid certificates for Google, which is a major risk given previous instances of CAs issuing false certificates, as discussed in Section 2.10.2. The entire setup process of an Android device was then demonstrated to be carried out over an unpinned connection as well, raising significant security and privacy concerns in the event that a man-in-the-middle attack is carried out with a valid CA-signed certificate in use. Thirdly, the setup process was shown to reveal, again over an unpinned HTTPS connection, plaintext user password candidates undergoing strength evaluation, therefore exposing every password a user attempts to use during the process of account creation. This again poses a concern in the event that the connection is being intercepted. The check-in process also was shown to be capable of remotely enabling device administrator functionality on the device, before login had even taken place. Finally, it was hypothesised and demonstrated to be accurate that the Android Device Manager service acts as a bypass to device encryption, allowing for Google to remotely change the lockscreen password on a device, which would also change the device encryption password. This poses significant concerns for users wishing to ensure the confidentiality of their data, and raised the concern that a compromise of Google's systems could result in widespread denial of service attacks against Android users.

These issues were reported to Google, which defended the behaviour of the system as by-design, but did not address the security concerns highlighted. The ability for an application with device administrator access to change the device lockscreen credentials, and thus encryption key, has since been removed from Android 7.0 and onwards [194].

5.3 Encryption Applications on Android

The Android platform commonly ships with the Google Play Store, allowing users to install third party software and, more recently, multimedia content to their Android device [6]. Previous work has explored the issue of malicious software being distributed on the Play Store, and the prevalence of it [195]. Other works have focused on the ability for forensic recovery of data from popular instant-messaging applications [196], and on the security of SSL implementation within applications [197].

This section shall explore a number of Android applications available on the Google Play Store, each of which claims to encrypt user data, to protect it from unauthorised access or theft, in order to enhance a user's privacy by giving them control over who may access their files. The purpose of this exercise was to evaluate how well-protected user data was, using existing encryption tools available on the platform, and to ascertain if these applications were protecting user data in the manner they claimed.

5.3.1 Selection of Applications and Testing Methodology

A range of applications were investigated, each of which had more than 5,000 users, according to the Google Play Store. Two had between 5 and 10 million users, and two had between 500,000 and 1 million users. Two apps were also by so-called Google Play "Top Developers". For balance, some less popular applications were also selected, with a few thousand users, in order to give a distribution of applications. Due to some applications having similar or identical names, each application is introduced with its package name (of the approximate form `com.x.y.z`), since this is a static identifier which uniquely defines the application in the Android ecosystem.

Each application was installed on a freshly-reset Motorola G 2014 Android device, running Android 5.0.2, the latest version of Android available for it at the time of carrying out the investigation. All applications were used in their default configuration. Typically, the application would prompt for the configuration of a PIN, and then the user would be able to use the application. Future launches of the application would require the entry of this PIN. Rather than attempt to attack or recover this PIN, the focus

[6] https://play.google.com

of this work was on the quality of encryption used to protect user files — the findings highlight that these PINs are immaterial to the security of the files, and therefore merely serve as "security-through-obscurity" to stop a casual user from accessing files.

For each application, a video or image file was captured using the device's camera, and was stored on the device's internal storage. Prior to encrypting it, it was retrieved to a computer using the Android Debugging Bridge (ADB) feature. The ADB interface was also used to use a Linux shell to explore the shared storage filesystem on the device, to locate the encrypted files, and to then retrieve them for analysis. The Android device used was not rooted, and elevated privileges were not used on the device during this investigation — at all times, only files from the shared storage partition of the device were retrieved, and these are files which are accessible to any application running on the Android device. The only exception to this was for ease of extraction of the database for the Password Locker application, as discussed in more detail in Section 5.3.5.

The following sections shall consider each of the applications, its security claims, and the findings of this work.

5.3.2 Vlocker-Hide Videos

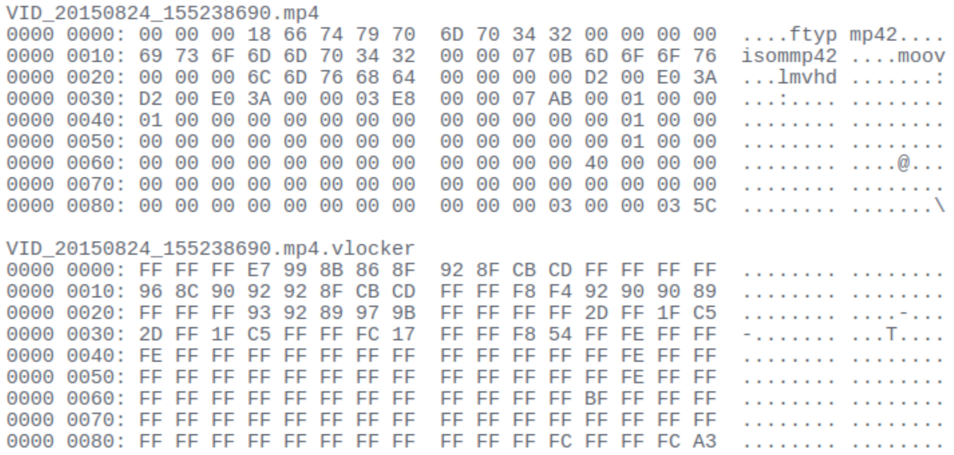
The Vlocker application used package name `com.simpleapp.vlocker` [198], and version 1.0.1 was investigated, which was the latest version available, released on 20th January 2016. At the time of writing, it had between 500,000 and 1 million downloads from the Play Store, and 4574 reviews with an average score of 4.2/5 stars.

5.3.2.1 Developer Claims

The developer of the software claimed to use encryption within the application — "Vlocker is the Super Video Hider. Lightning encryption and password recovery feature allow your privacy more secure". More specifically, the developers claim "Encryption - your videos are encrypted using advanced 128 bit AES encryption", although somewhat intriguingly also claim they allow for email-based recovery of a user's PIN within the application [198].

5.3.2.2 Operation of Application

Following use of the encryption feature of vlocker, the captured video file was no longer available at its original path. A hidden directory `.vlocker` was noted to have been added to the `/sdcard` shared storage area of the device. Within this folder was a folder layout of the *vault* shown by the application. A new file was found, with the same name as the original file, with the suffix `.vlocker` appended to it. This file was retrieved to



```
VID_20150824_155238690.mp4
0000 0000: 00 00 00 18 66 74 79 70  6D 70 34 32 00 00 00 00  ....ftyp mp42....
0000 0010: 69 73 6F 6D 6D 70 34 32  00 00 07 0B 6D 6F 6F 76  isommp42 ....moov
0000 0020: 00 00 00 6C 6D 76 68 64  00 00 00 00 D2 00 E0 3A  ...lmvhd .......:
0000 0030: D2 00 E0 3A 00 00 03 E8  00 00 07 AB 00 01 00 00  ...:.... ........
0000 0040: 01 00 00 00 00 00 00 00  00 00 00 00 00 01 00 00  ........ ........
0000 0050: 00 00 00 00 00 00 00 00  00 00 00 00 00 01 00 00  ........ ........
0000 0060: 00 00 00 00 00 00 00 00  00 00 00 00 40 00 00 00  ........ ....@...
0000 0070: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 0080: 00 00 00 00 00 00 00 00  00 00 00 03 00 00 03 5C  ........ .......\

VID_20150824_155238690.mp4.vlocker
0000 0000: FF FF FF E7 99 8B 86 8F  92 8F CB CD FF FF FF FF  ........ ........
0000 0010: 96 8C 90 92 92 8F CB CD  FF FF F8 F4 92 90 90 89  ........ ........
0000 0020: FF FF FF 93 92 89 97 9B  FF FF FF FF 2D FF 1F C5  ........ ....-...
0000 0030: 2D FF 1F C5 FF FF FC 17  FF FF F8 54 FF FE FF FF  -....... ...T....
0000 0040: FE FF FF FF FF FF FF FF  FF FF FF FF FF FE FF FF  ........ ........
0000 0050: FF FF FF FF FF FF FF FF  FF FF FF FF FF FE FF FF  ........ ........
0000 0060: FF FF FF FF FF FF FF FF  FF FF FF FF BF FF FF FF  ........ ........
0000 0070: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ........ ........
0000 0080: FF FF FF FF FF FF FF FF  FF FF FF FC FF FF FC A3  ........ ........
```

Figure 5.8
Comparison of original file and vlocker-protected file

the computer using ADB.

An initial inspection of the so-called encrypted file indicated that it had not been encrypted with AES-128, as claimed by the developers. Indeed, after the first 8192 bytes of the file, the remainder was byte-for-byte identical with the original file, indicating that only the MPEG4 header had been modified, as shown in Figure 5.8, which shows a comparison of the original file with the file after processing by vlocker. Cursory inspection highlights that this header has not been encrypted by AES, as the output distribution is nowhere approaching uniform — in particular, there is a high bias towards bytes having the value `0xFF`. Indeed, where a byte of `0x00` would be expected in the original video, a byte of `0xFF` was seen in the vlocker-protected file.

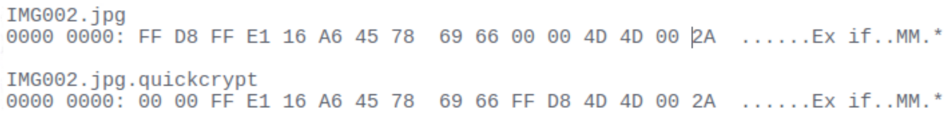
The remainder of the bytes of the header appeared to have simply had each of their bits individually flipped. For example, the fourth byte of the file (`0x18` == 00011000_2), was encoded to `0xE7 =` 11100111_2, indicating that a bitwise NOT function had been applied to each byte of the header.

Therefore, Equation 5.1a was formed, to determine the output of the encryption process. It can be reversed by re-arranging for decryption, as seen in Equation 5.1b.

$$ciphertext[i] = 255 - plaintext[i] \tag{5.1a}$$

$$plaintext[i] = 255 - ciphertext[i] \tag{5.1b}$$

Carrying out this process across the first 8192 bytes of the header revealed the original file, and this was confirmed by both per-byte comparison of the file, as well as the original and decoded files having the same cryptographic hash.



```
IMG002.jpg
0000 0000: FF D8 FF E1 16 A6 45 78  69 66 00 00 4D 4D 00 2A  ......Ex if..MM.*

IMG002.jpg.quickcrypt
0000 0000: 00 00 FF E1 16 A6 45 78  69 66 FF D8 4D 4D 00 2A  ......Ex if..MM.*
```

Figure 5.9
Comparison of original file against FotoX-protected file

5.3.2.3 Security Conclusions

Therefore, it can be concluded that, contrary to the claims of the developers, vlocker did not make use of AES-128 encryption. Indeed, it did not make use of encryption at all — the protection applied to files is merely that from hiding them in a folder whose name begins with a "." character, which typically hides them from view, and by inverting the bytes of the MPEG-4 header. This amounts to basic obfuscation, and does not protect user data as the application claims to.

5.3.3 Hide Pictures & Videos - FotoX

FotoX, available on the Play Store using package name `com.smsrobot.photox`, claims that "all your private data will be secured, encrypted and invisible to other Gallery apps" [199]. The developers, SMSROBOT Ltd, are listed as being "Top Developers" on the Google Play Store, and FotoX has between one and 5 million installs, with 21,131 reviews giving an average of 4.3/5 stars. Version 1.9 of the application was investigated, released in October 2015, the most recent available at the time of investigation in January 2016.

5.3.3.1 Operation of Application

FotoX was used to protect a JPEG image which was stored on the Android device's shared storage. Following the use of the encryption process, the file was no longer visible within the original directory. Like with vlocker, a hidden folder whose name began with dot was used to hide the folder from FotoX.

Within the directory `.FotoX` of the shared storage was the filesystem layout of the so-called *vault*, and these folders contained the protected files. The encrypted file had the suffix `.quickcrypt` appended to its name. It was extracted using ADB, and compared to the original file. This comparison indicated that only the very first few bytes of the file differed. Figure 5.9 shows a comparison of this region of the header of the file.

Specifically, FotoX had only swapped a pair of bytes — the first two bytes of the file, containing the JPEG magic bytes of `FF D8` had been swapped with the 11th and 12th bytes, which were `00 00`. This was the only difference between the two files, and reversal was trivial, by swapping the bytes back to their original locations, to obtain the original file.

5.3.3.2 Security Conclusions

It was clear that FotoX does not employ encryption in its handling of images, as was claimed in its description. The swapping of 2 bytes in the header with another 2 bytes was sufficient to ensure the image would not open in image viewers, but this does not offer any level of security as one would expect from software claiming to implement encryption.

Another application from the same developer was also investigated — "Vault - Hide Photos/App Lock", using package name `com.smsrobot.vault`, had 500,000 to one million downloads on the Play Store, and an average of 4.1/5 stars from 7249 reviews.

Like with FotoX, Vault claims that "Once in the Vault, all your private data will be secured, encrypted and invisible to other Gallery apps". Investigation revealed that Vault used the same process as described in this section to protect images, swapping bytes from the header. There was once again no encryption involved in protecting users' files, despite the developer's claims in the description of the application.

5.3.4 Video Locker & Photo Locker (Handy Apps)

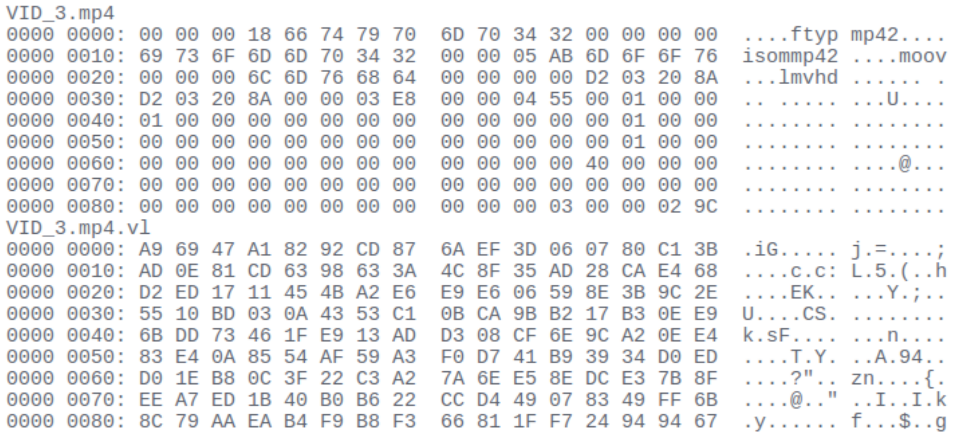
Video Locker is an application by the Google Play Store "Top Developer" Handy Apps. It had an average rating of 4.3/5 stars from 144,343 reviews, and had between 5 and 10 million installs. Version 1.2.1 was investigated, the latest available at the time, as of January 2016. Photo Locker is a similar application by the same developers, with an average rating of 4.2/5 stars, after 151,636 reviews. Photo Locker has between 10 and 50 million installs. Version 1.2.1 was again investigated.

5.3.4.1 Developer Claims

The developers of Video Locker claim it is "the ultimate secret gallery app", and that a key feature is its encryption, according to its Play Store description:

> "Encryption - hidden videos are not only moved to a secret location on your phone but are also encrypted using advanced 128 bit AES encryption. This means that even if someone manage to steal your SD card and copy the hidden video files, they will still be unable to view the locked videos" [200]

Identically worded claims were made for Photo Locker [201], including the assertion of 128-bit AES encryption. Since the operation of the two applications were found to



```
VID_3.mp4
0000 0000: 00 00 00 18 66 74 79 70  6D 70 34 32 00 00 00 00  ....ftyp mp42....
0000 0010: 69 73 6F 6D 6D 70 34 32  00 00 05 AB 6D 6F 6F 76  isommp42 ....moov
0000 0020: 00 00 00 6C 6D 76 68 64  00 00 00 00 D2 03 20 8A  ...lmvhd ...... .
0000 0030: D2 03 20 8A 00 00 03 E8  00 00 04 55 00 01 00 00  .. ..... ...U....
0000 0040: 01 00 00 00 00 00 00 00  00 00 00 00 00 01 00 00  ........ ........
0000 0050: 00 00 00 00 00 00 00 00  00 00 00 00 00 01 00 00  ........ ........
0000 0060: 00 00 00 00 00 00 00 00  00 00 00 00 40 00 00 00  ........ ....@...
0000 0070: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 0080: 00 00 00 00 00 00 00 00  00 00 00 03 00 00 02 9C  ........ ........
VID_3.mp4.vl
0000 0000: A9 69 47 A1 82 92 CD 87  6A EF 3D 06 07 80 C1 3B  .iG..... j.=....;
0000 0010: AD 0E 81 CD 63 98 63 3A  4C 8F 35 AD 28 CA E4 68  ....c.c: L.5.(..h
0000 0020: D2 ED 17 11 45 4B A2 E6  E9 E6 06 59 8E 3B 9C 2E  ....EK.. ...Y.;..
0000 0030: 55 10 BD 03 0A 43 53 C1  0B CA 9B B2 17 B3 0E E9  U....CS. ........
0000 0040: 6B DD 73 46 1F E9 13 AD  D3 08 CF 6E 9C A2 0E E4  k.sF.... ...n....
0000 0050: 83 E4 0A 85 54 AF 59 A3  F0 D7 41 B9 39 34 D0 ED  ....T.Y. ..A.94..
0000 0060: D0 1E B8 0C 3F 22 C3 A2  7A 6E E5 8E DC E3 7B 8F  ....?".. zn....{.
0000 0070: EE A7 ED 1B 40 B0 B6 22  CC D4 49 07 83 49 FF 6B  ....@.." ..I..I.k
0000 0080: 8C 79 AA EA B4 F9 B8 F3  66 81 1F F7 24 94 94 67  .y...... f...$..g
```

Figure 5.10
Comparison of original file against Video Locker-protected file

be near identical, the operation of the two shall be considered together — only minor differences exist between the applications, specifically around the types of file accepted by each application.

5.3.4.2 Operation of Application

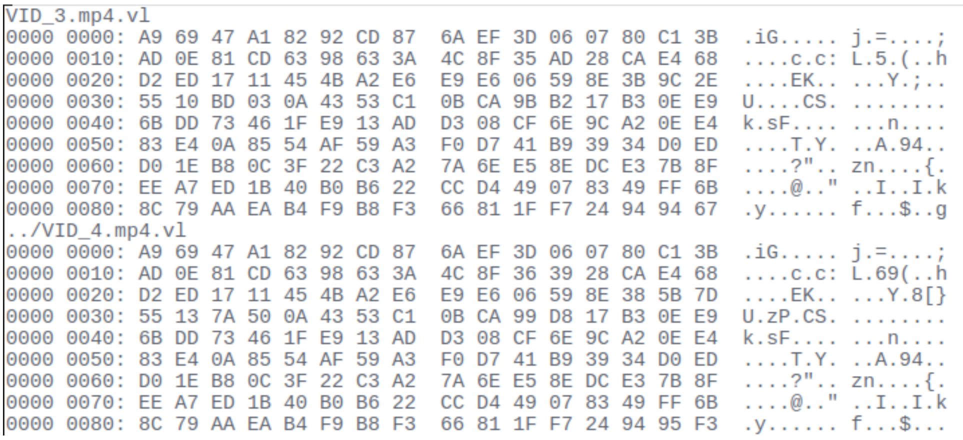
Upon encrypting a video with Video Locker, it was found to have been removed from its original location. A new folder within the shared storage, named `.VL` was identified, containing the vault contents. Each file had `.vl` appended to its name. Photo Locker created a similar directory named `.PL`.

For videos, only the first 8192 bytes differed from the original file, indicating that only the file header had been encrypted. Likwise for images, only the first 2048 bytes had been modified. It was noticed that the header contents appeared to be more uniformly distributed than the previous applications investigated. This indicated that a proper encryption algorithm may have been used, which would produce a ciphertext relatively indistinguishable from random data.

Figure 5.10 shows a comparison of the first 144 bytes of the original file, against the protected file, showing the seemingly-encrypted header data.

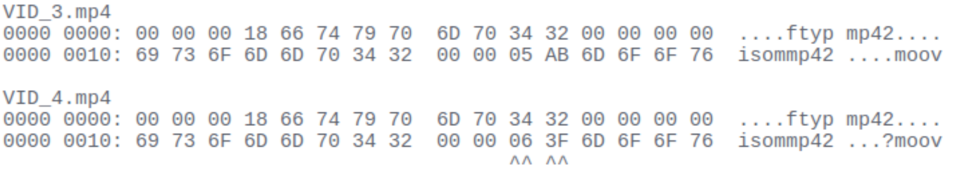
To verify if encryption was being properly applied, all Video Locker application data was fully erased from the device, including the vault. This meant that the application had lost all of its state data. Therefore, it believed it was running under a new install, and a new setup process was completed, with a different PIN used within the app. Another (different) video was then encrypted by Video Locker, and the encrypted video extracted.

The header of the two encrypted files were now compared — if these two ciphertexts held similarities, there would therefore be a correlation between the ciphertexts, indi-



```
VID_3.mp4.vl
0000 0000: A9 69 47 A1 82 92 CD 87  6A EF 3D 06 07 80 C1 3B  .iG..... j.=....;
0000 0010: AD 0E 81 CD 63 98 63 3A  4C 8F 35 AD 28 CA E4 68  ....c.c: L.5.(..h
0000 0020: D2 ED 17 11 45 4B A2 E6  E9 E6 06 59 8E 3B 9C 2E  ....EK.. ...Y.;..
0000 0030: 55 10 BD 03 0A 43 53 C1  0B CA 9B B2 17 B3 0E E9  U....CS. ........
0000 0040: 6B DD 73 46 1F E9 13 AD  D3 08 CF 6E 9C A2 0E E4  k.sF.... ...n....
0000 0050: 83 E4 0A 85 54 AF 59 A3  F0 D7 41 B9 39 34 D0 ED  ....T.Y. ..A.94..
0000 0060: D0 1E B8 0C 3F 22 C3 A2  7A 6E E5 8E DC E3 7B 8F  ....?".. zn....{.
0000 0070: EE A7 ED 1B 40 B0 B6 22  CC D4 49 07 83 49 FF 6B  ....@.." ..I..I.k
0000 0080: 8C 79 AA EA B4 F9 B8 F3  66 81 1F F7 24 94 94 67  .y...... f...$..g
../VID_4.mp4.vl
0000 0000: A9 69 47 A1 82 92 CD 87  6A EF 3D 06 07 80 C1 3B  .iG..... j.=....;
0000 0010: AD 0E 81 CD 63 98 63 3A  4C 8F 36 39 28 CA E4 68  ....c.c: L.69(..h
0000 0020: D2 ED 17 11 45 4B A2 E6  E9 E6 06 59 8E 38 5B 7D  ....EK.. ...Y.8[}
0000 0030: 55 13 7A 50 0A 43 53 C1  0B CA 99 D8 17 B3 0E E9  U.zP.CS. ........
0000 0040: 6B DD 73 46 1F E9 13 AD  D3 08 CF 6E 9C A2 0E E4  k.sF.... ...n....
0000 0050: 83 E4 0A 85 54 AF 59 A3  F0 D7 41 B9 39 34 D0 ED  ....T.Y. ..A.94..
0000 0060: D0 1E B8 0C 3F 22 C3 A2  7A 6E E5 8E DC E3 7B 8F  ....?".. zn....{.
0000 0070: EE A7 ED 1B 40 B0 B6 22  CC D4 49 07 83 49 FF 6B  ....@.." ..I..I.k
0000 0080: 8C 79 AA EA B4 F9 B8 F3  66 81 1F F7 24 94 95 F3  .y...... f...$...
```

Figure 5.11
Comparison of two different Video Locker-protected files



```
VID_3.mp4
0000 0000: 00 00 00 18 66 74 79 70  6D 70 34 32 00 00 00 00  ....ftyp mp42....
0000 0010: 69 73 6F 6D 6D 70 34 32  00 00 05 AB 6D 6F 6F 76  isommp42 ....moov

VID_4.mp4
0000 0000: 00 00 00 18 66 74 79 70  6D 70 34 32 00 00 00 00  ....ftyp mp42....
0000 0010: 69 73 6F 6D 6D 70 34 32  00 00 06 3F 6D 6F 6F 76  isommp42 ...?moov
                                          ^^ ^^
```

Figure 5.12
Comparison of the original video files

cating that similar plaintexts resulted in similar ciphertexts. As shown in Figure 5.11, the two headers were similar for the vast majority of bytes, indicating that this was likely use of a statically initialised cipher. Indeed, the offsets of the bytes differing in the ciphertext were the same offsets as the bytes differing in the plaintexts. For example, at offset `0x1A` of Figure 5.11, two bytes differ between the ciphertexts.

Figure 5.12 shows that these same bytes in the plaintext differed. For ease of identification, those bytes are highlighted with arrows.

Therefore, it was clear that, given the correlation between the two ciphertexts, the same key and initialisation parameters were being used, even though a different PIN was being used in the application for the encryption of the second file. This was confirmed by using the XOR function across differing bytes of both plaintext and ciphertext. For example, from Figure 5.12, bytes `0x31 - 0x33` were $[03, 20, 8A]$ in VID_3, and $[00, E7, D9]$ in VID_4. Within the ciphertexts, these bytes were $[10, BD, 03]$ and $[13, 7A, 50]$ respectively.

By carrying out the XOR function across the plaintexts, the result was $[00 \oplus 03, 20 \oplus E7, 8A \oplus D9] = [03, C7, 53]$. Across the ciphertexts, the result of XOR was $[10 \oplus 13, BD \oplus 7A, 03 \oplus 50] = [03, C7, 53]$, showing that the XOR of the two ciphertexts revealed the XOR of the two plaintexts, thus proving the same key and cipher parameters are used on different files, thus leaking information about the difference between different ciphertexts, and suggesting the use of a stream cipher mode of operation.

Since the same key was used for all data encrypted by the app, and this was static between different installations of the app, this makes the ciphertext vulnerable to a simple known-plaintext attack. By encrypting a large file, and recording both the plaintext and ciphertext of that file, any arbitrary file may be decrypted with the XOR function, since an unknown ciphertext was able to be XOR'd with the ciphertext whose corresponding plaintext is known, and the result XOR'd with the known plaintext to reveal the unknown plaintext. Alternatively, a ciphertext-only attack could be carried out, since the XOR of two bytes of ciphertext corresponds with the XOR of the corresponding two plaintext bytes. This conveys the XOR distance between the two bytes. By then XOR'ing this distance with known dictionary values or header bytes, and moving this candidate data around the file, a match may be found yielding the second plaintext, since the XOR of the two ciphertexts is equivalent to the XOR of the two plaintexts.

5.3.4.3 Decryption of Video Locker & Photo Locker Data

While the above information indicated that that files could be decrypted by an attacker, the process used to protect user files remained somewhat unclear and convoluted. It was identified firstly that using a different PIN and recovery email address for the application did not affect the encryption procedure — the same ciphertext was generated in each case. This also confirmed the static nature of the IV and key, and confirmed the above findings carried across different PINs and recovery email addresses.

A file named `.config` was located within the root directory of both apps' vaults. This file contained two base64-encoded strings, which themselves decoded to scrambled data. By analysing the operation of the application upon a clean install, and after restoring this data, it emerged that this information was held to allow a user to transfer their data to a new Android device. This file contained a protected copy of the user's PIN and recovery email address. Neither was padded, allowing for trivial identification of the length of each — following decoding from base64, the number of bytes was the same as the length of each string.

Since the recovery process could be initiated on a new device, it was clear this data must be able to be accessed by the application itself, and it was clear the user PIN was not hashed, given its length. It was determined that a static key was used to decrypt this data, using AES-128 in CTR mode, with a static (fixed) IV. Analysis of the strings within the application binary revealed that the key and IV were constant, static values which were stored within the application.

5.3.4.4 Security Conclusions

While Video Locker does indeed use encryption, it only encrypted the header of the MPEG-4 file, rather than its contents. It used static parameters for this header encryption, for all files, irrespective of PIN used, leaking information between files. These static parameters were hard-coded into the application and therefore offer no security to users. Indeed, the use of CTR encryption with fixed initialisation vector also leaked other information, although the use of a static encryption key and IV meant that anyone with access to this widely-used software, or knowledge of how it operates, may decrypt files by any other user. It therefore offers no realistic level of security against third parties wishing to access user data.

5.3.5 Password Locker

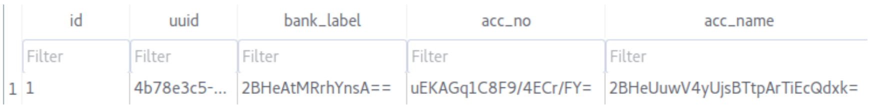
Password Locker is an application by Handy Apps, the same developer as Video Locker and Photo Locker, and also a Google Play Store "Top Developer". It has between 100,000 and 500,000 users, and an average rating of 3.9/5 from 1,179 reviews. Password Locker, as the name suggests, is designed to securely store user passwords, which are naturally highly sensitive.

For example, the developers state that Password Locker "stores your sensitive information offline and passwords safe, secure and organised", and that it has "many optional convenient features for it to be the best password manager ever designed specifically for Android" [202]. Specific details are also given as to the encryption supposedly used — the developers state:

> *Secure*
> Lock up your data in Password Locker with extremely tough and strong 256-bit AES encryption - military level encryption (takes trillions of years to decrypt) [202]

5.3.5.1 Operation of Application

Password Locker stored its password database within the application's private storage, meaning that root access was required to retrieve the database for the purpose of this analysis. Note, however, that it also offers a paid feature to enable cloud synchronisation of passwords with a user's Dropbox or Google Drive account, which may expose this database to third party services. Nonetheless, given the ease with which Android devices may be rooted with exploits such as CVE-2014-3153 (TowelRoot) and CVE-2015-3636 (Ping Sockets root), it is possible that a malicious party may gain access to the database through an application on the device. Nonetheless, if the claims made by the developers



	id	uuid	bank_label	acc_no	acc_name
	Filter	Filter	Filter	Filter	Filter
1	1	4b78e3c5-...	2BHeAtMRrhYnsA==	uEKAGq1C8F9/4ECr/FY=	2BHeUuwV4yUjsBTtpArTiEcQdxk=

Figure 5.13
Password Locker Database Structure

are accurate, users would have nothing to fear, as their data would be appropriately encrypted.

The database was found to hold base64-encoded fields, in the structure of the records, as shown in Figure 5.13. From this, it was immediately possible to identify the use of weak, potentially broken cryptography, by the observation of a common prefix between the two *ciphertexts* for `bank_label` and `acc_name`. Since this was simply the default record created by the application, it was possible to verify the hypothesis that they shared a prefix of 3 characters (given the 4 base64-encoded characters in common). Indeed, this suspicion was found correct — the bank account label was “Sam Sample” and the account name was “Sample Checking Acct”. Once again, there was no padding present in the ciphertexts, and lengths of plaintexts could be identified directly from ciphertext lengths.

The presence of prefixes also indicated that the cipher in use was not being initialised with unique parameters for each operation. Therefore, to demonstrate the ability for key recovery, the following process was carried out, where kc is a known ciphertext, kp is the corresponding known plaintext, and uc is a ciphertext with an unknown corresponding plaintext, with the length of $uc < kc$:

$$key = kc \oplus kp \tag{5.2a}$$

$$up = key \oplus uc \tag{5.2b}$$

By XOR’ing a known plaintext and ciphertext together per Equation 5.2a, the AES block key is obtained. By then XOR’ing this block key against a ciphertext with unknown plaintext, for the length of this ciphertext, discarding any remaining block key material, the unknown plaintext up was recovered, per Equation 5.2b.

This was confirmed across 2 Android devices, with different PINs and security parameters set on each, to prove that the key used is static, and not derived from the user’s password. Therefore, the data is effectively only obfuscated. Anyone carrying out the above may decrypt any other user’s Password Locker database trivially, using Equation 5.2b, since key is constant and hard-coded across all installations of the application. Once again, AES was being used in CTR mode.

5.3.5.2 Security Conclusions

Password Locker's improper use of encryption raises significant concerns, specifically for users synchronising their data to an external service, as permitted by the application. It was identified that the same hard-coded AES key was used between different installs of the application, thus rendering the encryption ineffective, and merely obfuscation. Given the importance of passwords, and the perception of security offered here, users may be at risk, in the event their password database was able to be retrieved by an attacker.

5.3.6 Video Locker (NewSoftwares.net)

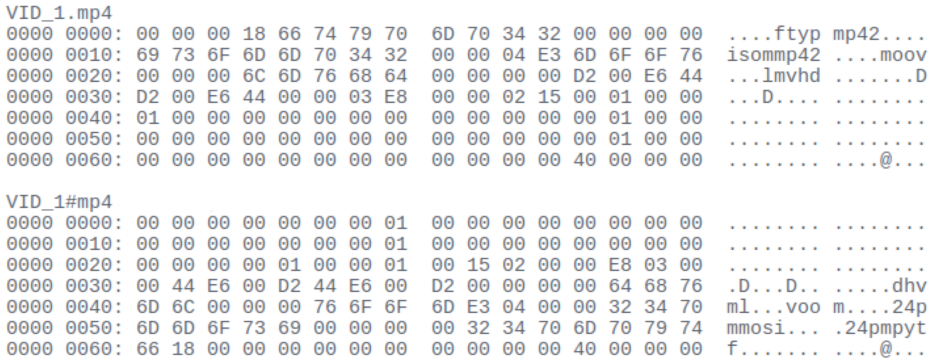
Video Locker is an application by the developer NewSoftwares.net. To avoid ambiguity with the other application named Video Locker, investigated previously, this shall refer to this application as Video Locker Advanced, in-keeping with its package name. It had an average rating of 4.2/5 stars from 192 reviews, and had between 10,000 and 50,000 installs. Version 1.0.3 of the application was investigated, which was released in January 2016, and the latest available. The developers claim to use "Encryption - The app locks your personal videos, prevents video hack.", and that it protects private videos "using fast encryption techniques" [203].

5.3.6.1 Operation of Application

Video Locker Advanced was used to encrypt a video captured from the camera on the test phone. The video was retrieved from the device prior to its encryption to provide a comparison.

After encryptionr, and in-keeping with the other apps investigated so far, the file was no longer visible in its original location. A new directory (which was not hidden) was located within the root folder of the device share storage, titled `Video Locker Advanced Encrypted Data`. Within this directory was a vault structure, and the encrypted file was located, with the original file extension separator dot replaced with the `#` symbol. Therefore a file named `VID_1.mp4` became `VID_1#mp4`.

Comparison of the original file with the protected file indicated that only the header of the video had been modified, with the first 100 bytes of the header changed in position. Therefore, the fourth byte became the 96th byte, as shown in Figure 5.14 — the fourth byte `0x18` is seen at address `0x61`. The ASCII representation makes this reversal of the bytes clearer, as shown in the right column of Figure 5.14.



```
VID_1.mp4
0000 0000: 00 00 00 18 66 74 79 70  6D 70 34 32 00 00 00 00  ....ftyp mp42....
0000 0010: 69 73 6F 6D 6D 70 34 32  00 00 04 E3 6D 6F 6F 76  isommp42 ....moov
0000 0020: 00 00 00 6C 6D 76 68 64  00 00 00 00 D2 00 E6 44  ...lmvhd .......D
0000 0030: D2 00 E6 44 00 00 03 E8  00 00 02 15 00 01 00 00  ...D.... ........
0000 0040: 01 00 00 00 00 00 00 00  00 00 00 00 00 01 00 00  ........ ........
0000 0050: 00 00 00 00 00 00 00 00  00 00 00 00 00 01 00 00  ........ ........
0000 0060: 00 00 00 00 00 00 00 00  00 00 00 00 40 00 00 00  ........ ....@...

VID_1#mp4
0000 0000: 00 00 00 00 00 00 00 01  00 00 00 00 00 00 00 00  ........ ........
0000 0010: 00 00 00 00 00 00 00 01  00 00 00 00 00 00 00 00  ........ ........
0000 0020: 00 00 00 00 01 00 00 01  00 15 02 00 00 E8 03 00  ........ ........
0000 0030: 00 44 E6 00 D2 44 E6 00  D2 00 00 00 00 64 68 76  .D...D.. .....dhv
0000 0040: 6D 6C 00 00 00 76 6F 6F  6D E3 04 00 00 32 34 70  ml...voo m....24p
0000 0050: 6D 6D 6F 73 69 00 00 00  00 32 34 70 6D 70 79 74  mmosi... .24pmpyt
0000 0060: 66 18 00 00 00 00 00 00  00 00 00 00 40 00 00 00  f....... ....@...
```

Figure 5.14
Comparison of original file against Video Locker-protected file

5.3.6.2 Security Conclusions

From the above, it was clear that Video Locker Advanced did not use the advanced encryption techniques which it claimed — this amounted to reversing the bytes of the file header. Concerningly, the application features Dropbox backup support [203], which may lead users to believe that they are uploading only encrypted data to Dropbox, when they are in fact uploading plaintext user files with merely minor obfuscation of the file headers.

5.3.7 Gallery Vault

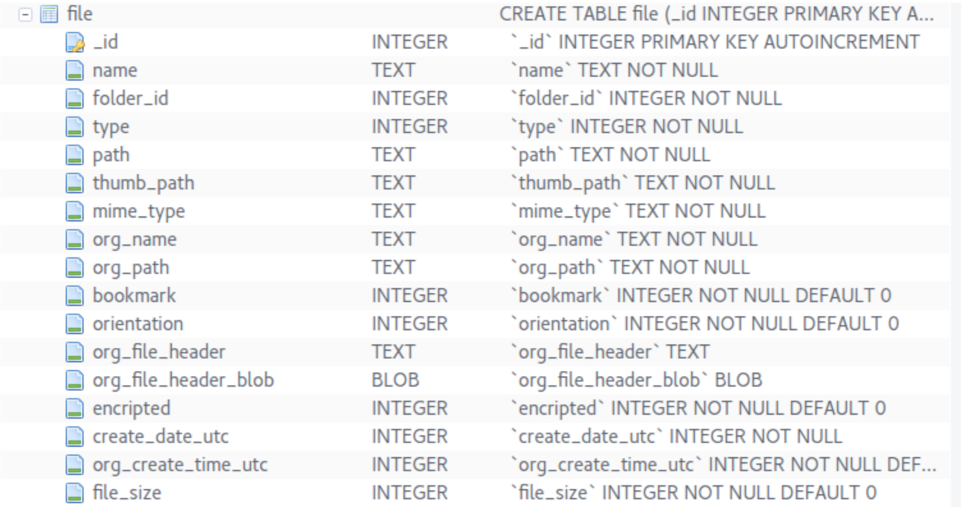
Gallery Vault is an application by ThinkYeah Mobile, with between 10 and 50 million reported users on the Play Store. It has an average rating of 4.4/5, based on 223,160 reviews. Version 2.6.5 was investigated, which was the latest version available as of January 2016.

The developers state that "The hidden file are all encrypted", and that "GalleryVault is a fantastic privacy protection app to easily hide and encrypt your photos, videos and any other files that you do not want others to see" [204].

5.3.7.1 Operation of Application

Like the other applications investigated, Gallery Vault created its own vault area on the shared storage, under the directory name `.galleryvault_DoNotDelete_X`, where X was the Unix epoch time in seconds of the creation of the vault.

Encrypted files were stored within a directory named `file`, and named after the epoch time of their encryption. While this appeared initially to hide the filenames, a folder named `backup` was located adjacent to the vault, containing a backup of the application's internal database, `galleryvault.db`. Note that this database was contained within the device shared storage, and was therefore accessible to any software on the



file		CREATE TABLE file (_id INTEGER PRIMARY KEY A...
_id	INTEGER	`_id` INTEGER PRIMARY KEY AUTOINCREMENT
name	TEXT	`name` TEXT NOT NULL
folder_id	INTEGER	`folder_id` INTEGER NOT NULL
type	INTEGER	`type` INTEGER NOT NULL
path	TEXT	`path` TEXT NOT NULL
thumb_path	TEXT	`thumb_path` TEXT NOT NULL
mime_type	TEXT	`mime_type` TEXT NOT NULL
org_name	TEXT	`org_name` TEXT NOT NULL
org_path	TEXT	`org_path` TEXT NOT NULL
bookmark	INTEGER	`bookmark` INTEGER NOT NULL DEFAULT 0
orientation	INTEGER	`orientation` INTEGER NOT NULL DEFAULT 0
org_file_header	TEXT	`org_file_header` TEXT
org_file_header_blob	BLOB	`org_file_header_blob` BLOB
encripted	INTEGER	`encripted` INTEGER NOT NULL DEFAULT 0
create_date_utc	INTEGER	`create_date_utc` INTEGER NOT NULL
org_create_time_utc	INTEGER	`org_create_time_utc` INTEGER NOT NULL DEF...
file_size	INTEGER	`file_size` INTEGER NOT NULL DEFAULT 0

Figure 5.15
Schema of the GalleryVault database file table

```
IMG_20150825_135302229.jpg
0000 0000: FF D8 FF E1 3A AC 45 78  69 66 00 00 4D 4D 00 2A  ....:.Ex if..MM.*

1440507185556
0000 0000: 00 00 00 00 00 00 00 00  00 00 00 00 4D 4D 00 2A  ........ ....MM.*
```

Figure 5.16
Comparison of original file against GalleryVault-protected file

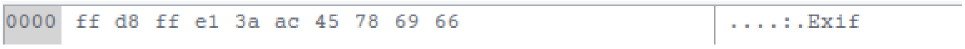
phone, and to anyone with access to the device.

This database contained a table named `file`, which stored an unencrypted mapping between protected and unprotected files. The field `org_name` and `org_path` contained the original name and path of the file respectively, with the path also including the original filename. The database schema is shown in Figure 5.15.

A JPEG photograph was taken, and encrypted using GalleryVault. After extracting the encrypted file from the vault, it was compared to the original file. Figure 5.16 shows this comparison — only the first ten bytes of the file were found to differ, and had simply been set to have byte values of zero.

While recovering from this would be straightforward, only requiring identification of the correct header values, based upon the image dimensions, it was found that this was not necessary, on account of the leakage of the original header information within the GalleryVault database file.

Within the file table, the field `org_file_header_blob` contained the plaintext original file header, as shown in Figure 5.17. Therefore, with access to only the GalleryVault-protected file, and the backup database held in an adjacent directory, within the globally accessible shared storage, it was possible to immediately recover the file header, which can be compared against Figure 5.16 to be identical to the original file's header which was removed.



```
0000  ff d8 ff e1 3a ac 45 78 69 66                    ....:.Exif
```

Figure 5.17
GalleryVault database leaking original file header

5.3.7.2 Security Conclusions

From the above, it is clear that GalleryVault did not carry out encryption of user image files. The first ten bytes of the file header were zeroed out, although the header was backed up, in plaintext, within an SQLite3 database that was held adjacent to the protected file. Therefore, anyone with access to the device can trivially restore these ten bytes, and have restored the original file.

5.3.8 Encrypt File Free

Encrypt File Free, by MobilDev, was the second application listed in the Play Store search for the query "encrypt". It had 50,000 to 100,000 installs, and an average rating of 3.6/5 from 255 reviews. Version 1.0.8 of the application, from November 2014, was the latest version available, and the version investigated.

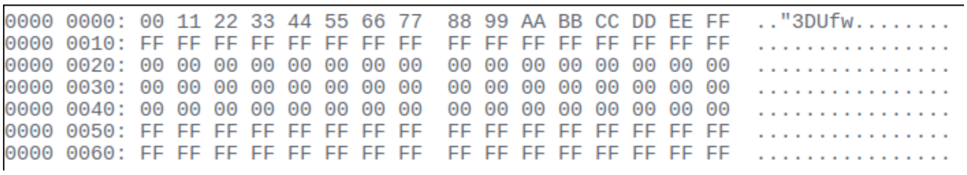
5.3.8.1 Developer Claims

The developer of Encrypt File Free states "Encrypt File Free can encrypt and protect photos, videos, audios, pictures, doc, ppt, xls, pdf and other files using a password", and that "The encrypted file can only be opened with the correct password" [205]. They also state users should "Encrypt your files and not just hide them. This solution is better and safer than simply hiding files".

5.3.8.2 Operation of Application

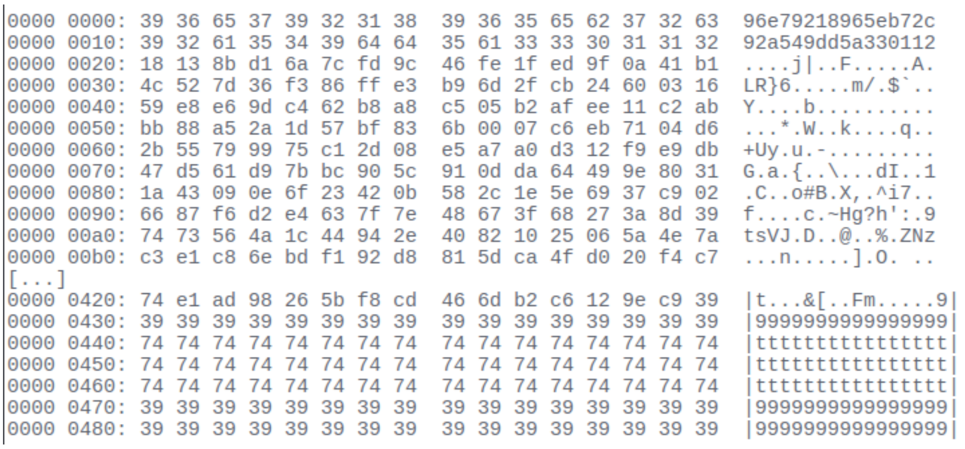
Since this tool was designed to encrypt files of any type, rather than specifically videos or images, a test file was created, as a first test of the algorithm. The test file consisted of the sequence of 16 increasing bytes, `00, 11, 22... FF`, followed by 16 bytes set to `FF`, 48 zero bytes, and a further 32 bytes set to `FF`. The intention of using this test file was to ascertain if the output of the cipher was strong and uniform, or weak and potentially breakable. The plaintext data is shown in Figure 5.18.

Upon encrypting this file, which contained a total of 112 bytes, a ciphertext of 1168 bytes was returned. Examination of this file highlighted that this file could likely be split into three chunks — an ASCII representation of a 16 bytes hex string, perhaps a hash like MD5, some unknown data, and finally data which appeared to be the encrypted content of the file. Figure 5.19 shows the resulting ciphertext output from Encrypt File



```
0000 0000: 00 11 22 33 44 55 66 77  88 99 AA BB CC DD EE FF  .."3DUfw........
0000 0010: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
0000 0020: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000 0030: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000 0040: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000 0050: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
0000 0060: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
```

Figure 5.18
Plaintext selected for testing of Encrypt File Free



```
0000 0000: 39 36 65 37 39 32 31 38  39 36 35 65 62 37 32 63  96e79218965eb72c
0000 0010: 39 32 61 35 34 39 64 64  35 61 33 33 30 31 31 32  92a549dd5a330112
0000 0020: 18 13 8b d1 6a 7c fd 9c  46 fe 1f ed 9f 0a 41 b1  ....j|..F.....A.
0000 0030: 4c 52 7d 36 f3 86 ff e3  b9 6d 2f cb 24 60 03 16  LR}6.....m/.$`..
0000 0040: 59 e8 e6 9d c4 62 b8 a8  c5 05 b2 af ee 11 c2 ab  Y....b..........
0000 0050: bb 88 a5 2a 1d 57 bf 83  6b 00 07 c6 eb 71 04 d6  ...*.W..k....q..
0000 0060: 2b 55 79 99 75 c1 2d 08  e5 a7 a0 d3 12 f9 e9 db  +Uy.u.-.........
0000 0070: 47 d5 61 d9 7b bc 90 5c  91 0d da 64 49 9e 80 31  G.a.{..\...dI..1
0000 0080: 1a 43 09 0e 6f 23 42 0b  58 2c 1e 5e 69 37 c9 02  .C..o#B.X,.^i7..
0000 0090: 66 87 f6 d2 e4 63 7f 7e  48 67 3f 68 27 3a 8d 39  f....c.~Hg?h':.9
0000 00a0: 74 73 56 4a 1c 44 94 2e  40 82 10 25 06 5a 4e 7a  tsVJ.D..@..%.ZNz
0000 00b0: c3 e1 c8 6e bd f1 92 d8  81 5d ca 4f d0 20 f4 c7  ...n.....].O. ..
[...]
0000 0420: 74 e1 ad 98 26 5b f8 cd  46 6d b2 c6 12 9e c9 39  |t...&[..Fm.....9|
0000 0430: 39 39 39 39 39 39 39 39  39 39 39 39 39 39 39 39  |9999999999999999|
0000 0440: 74 74 74 74 74 74 74 74  74 74 74 74 74 74 74 74  |tttttttttttttttt|
0000 0450: 74 74 74 74 74 74 74 74  74 74 74 74 74 74 74 74  |tttttttttttttttt|
0000 0460: 74 74 74 74 74 74 74 74  74 74 74 74 74 74 74 74  |tttttttttttttttt|
0000 0470: 39 39 39 39 39 39 39 39  39 39 39 39 39 39 39 39  |9999999999999999|
0000 0480: 39 39 39 39 39 39 39 39  39 39 39 39 39 39 39 39  |9999999999999999|
```

Figure 5.19
Ciphertext output from Encrypt File Free for above test file

Free, with some of the 1024-byte header truncated for readability.

The first 32 bytes were found to contain an ASCII representation of the plain, unsalted, MD5 hash of the user's PIN for the application. For the example shown in Figure 5.19, the PIN used was "111111" (as ASCII characters), and the MD5 hash shown in the first 32 bytes is an ASCII representation of $md5(111111)$. Therefore, the PIN was trivially exposed to anyone with access to a ciphertext produced by the application, on account of the ease of brute-forcing MD5 hashes. This may be a risk where users unintentionally expose their device or other PINs, as a result of the re-use of that PIN within this application. Also of interest was that if the original data and hash lengths $(112 + 32)$ were subtracted from the overall ciphertext length (1168), this indicated the middle section of the data occupied exactly 1024 bytes, perhaps suggesting padding or some form of fixed-length lookup table.

Indeed, by altering the PIN hash located at the header of the file, the same file could be decrypted by another device, which had never been in contact with the plaintext file, thus showing that the file was not being encrypted with the user PIN, and that its presence was merely for checking validity of the entered PIN. Therefore, the process was no better than storing the file in plaintext, since it could be trivially decoded without knowledge of the user's PIN. In contrast to the other applications investigated however, Encrypt File Free did actually modify the body of the file, rather than merely the headers, although it does not offer any effective security.

5.3.8.3 Cryptanalysis of the Output

The output of the cipher was found to be very weak, and appeared to represent that of a monoalphabetic substitution cipher. Specifically, the pattern of blocks of data was visible in the output, with 16 differing values, then 16 values (say A), then 48 different values (say B), then a further 32 bytes of A. This pattern indicated that the structure of the input plaintext was remaining constant through to the ciphertext. This can be seen at the lower part of Figure 5.19, where the pattern of repeated bytes has been exposed from the plaintext through to the ciphertext.

By focusing on the 1024-byte block of unknown header data, it was observed that different byte values appeared with slightly different frequencies. An entropy estimation by the Unix `ent` utility indicated that the header entropy was approximately 7.47 bits per byte, although with the arithmetic mean of data bytes significantly lower than expected, at around 80. Were the data uniformly distibuted, this would be expected to be nearer 127.5. The auto-correlation coefficient across the header was also somewhat elevated, at around 0.3, rather than 0, which would be expected for random and unpredictable data.

An inconsistency within the distribution of the data within the 1024-byte header was also noticed, since each byte value from `0x00` to `0xFF` was found to exist exactly once within the first 256 bytes. This therefore appeared to be a form of one-to-one look-up table, given the lack of duplicates, and presence of each value.

By using a crib from a known plaintext and ciphertext mapping, and the guess that a form of monoalphabetic substitution was taking place, it is possible to consider that the plaintext byte `0x00` from the first byte of the plaintext from Figure 5.18 was mapped to a ciphertext byte of `0x74`, per Figure 5.19. By observing that the byte `0x74` appears at offset `0x80` of this 256-byte header, it appears that the plaintext is obtained by subtracting the value `0x80`. This was verified for other byte values — the ciphertext byte `0x98` corresponded to plaintext `0x33`, and the ciphertext byte appeared at offset `0xb3`. Subtracting `0x80` from this resulted in the plaintext byte `0x33` as expected.

Therefore, it is possible to decode any arbitrary file protected by this application, simply through cryptanalysis of the ciphertext, and knowledge of a single plaintext created with the application. While the header varied between uses of the program, this process can be used to decode any file created by the application.

5.3.8.4 Security Conclusions

It has been shown that Encrypt File Free utilises weak obfuscation, which does not require knowledge of a key to access their so-called "encrypted" files. The cipher is effectively a monoalphabetic substitution cipher, and offers no protection from frequency analysis, with the mapping from plaintext to ciphertext being one-to-one. The process of identifying this and carrying out the attack was demonstrated, and shown to be able to be identified merely by analysing ciphertext output against a single known plaintext.

A written review from a user in September 2015 entitles "Unbreakable" states "Encryption still holds, after 2 months. (I hope) LoL" [205]. This does not appear to be the case, and the ciphertexts were not able to stand up to basic analysis, with a security level commensurate with that of a monoalphabetic substitution cipher. The application also leaked the unsalted, plain MD5 hash of the user's PIN in the header of each file, potentially exposing a user's PIN to other applications, which may be damaging if this were to be re-used in other scenarios, such as on a lock-screen or a bank card.

5.3.9 Conclusions of Analysis and Discussion

An analysis of a range of Android applications claiming to implement encryption to protect user files from unauthorised access was conducted. This selection of applications encompassed those from 5,000 to 10,000 users, through to those with 10 to 50 million users. These applications were highly rated, and in some cases are from "Top Developers" on the Google Play Store. Every application here claimed to encrypt files for privacy or security, although only one actually implemented a standard cipher, and it was trivially breakable and improperly implemented. In most of the other cases, applications merely obfuscated or removed file headers. In one case, the header was simply moved to a plaintext field within an SQLite3 database, stored next to the protected file.

The second application appearing on a Google Play Store search for "encrypt" was found to use a simple obfuscation algoithm to carry out so-called encryption of files, and a thorough an analysis of it has been presented, describing how to decrypt files which were "encrypted" using it. One application was also identified, which claimed to use AES-128 encryption, yet merely inverted the first 8192 bytes of the file header. Another application making such claims used AES-128 in CTR mode, with fixed key and initialisation vector, irrespective of a user's PIN, and it was demonstrated that XOR'ing two ciphertexts together would reveal the differences between the two plaintexts, indicating the inadequacy of this configuration. The static parameters were also extracted, allowing for the decryption of the file headers.

These findings are highly concerning, and show that there are significant security implications for users relying on software such as that which was investigated, particularly when applications have in excess of 10 million users reported by Google Play. These applications all claimed to use encryption, and a viable attack to recover plaintext from all of them has been demonstrated. Were users to rely on these applications for privacy, or to protect sensitive data, it may be recovered by an attacker carrying out analysis of the files and the applications concerned. None of these applications offered sufficient protection for user data, and none of them made use of any kind of "key" to alter the mapping of plaintext to ciphertext, a key requirement of an encryption algorithm [59, Section 1.4].

This also highlights an important consideration for users, which is that they are forced to trust software to do what it claims to. All of these applications claimed to encrypt files, and some even made specific claims as to the type of encryption used. Despite this, none of the applications investigated offered satisfactory security. Since none of these applications were open sourced, which is an important principle of Kerchoff's Assumption [206], where the algorithm and operation of software should be assumed to be known by an adversary. Since these applications did not do so, they did not encourage the auditing of their implementations, potentially leading to many millions of users misplacing their trust in this software. Indeed, obscurity offered no security here, as these applications were nonetheless breakable.

This highlights the important of open source, peer-reviewed, robust use of strong cryptography, specifically around the use of well-established algorithms correctly. None of the applications considered here gave any consideration to the need for authentication of ciphertexts, nor to the importance of ensuring no information about a plaintext was revealed in the ciphertext, such as structure or even full plaintext content, for the solutions only obfuscating file headers.

With all of the encryption apps considered within this work being relatively user friendly and user-focused, this gives rise to consideration as to the trade-offs between security and usability. Many of the applications considered here featured *password reset* functionality, allowing users to reset their encryption password if they forgot it. This naturally raises questions as to the level of security offered, if it is possible for the password to be easily reset by the user receiving an email. There does however raise a more general question, around whether or not it is ethical or appropriate to advertise software as being secure, when it is heavily vulnerable to attacks such as those demonstrated here. Were users to depend on this software for confidentiality, then suffer as a result of their data being accessed, despite being encrypted, there is a question around whether or not such descriptions were misleading or inaccurate. Given that the applications considered here were often not implementing any kind of encryption, their claims are clearly questionable at best.

The United States' FTC issues advice for app developers, encouraging them to consider security at the start of making an application [207], although much of this advice focuses more on privacy and data protection, rather than on proper implementation of cryptography. App developers have previously been found guilty of misleading practices, although these have typically focused on non-transparent fees to use applications, such as by sending premium rate SMS messages without making users aware [208].

It does appear to remain an open question, however, as to whether or not there is a legal case for claims of false or misleading advertising against mobile application developers, especially where an application is made available for free. While legislation exists to protect consumers from digital content sales [209], the rise in alternative business models, whereby the user does not pay directly for the application, but the developer receives money as a result of advertisements shown within the application to users, raises questions as to whether there is any recourse available for users against misleading claims made by developers.

5.4 Conclusions

This chapter has presented research carried out into the security of one component of the Android platform's defences against attacks on the network connection, as well as an analysis of a range of widely used Android encryption apps. This highlighted that the measures in place to protect the security of data being transmitted from an Android device to Google were inadequate, and demonstrated how this could be used to prevent the update of certificate pins to devices, thus breaking a key component of the network security layer. In addition, it was shown that the device setup process permitted the remote server to exert significant control over the client device, with the user's Android device inherently trusting the server with the plaintext of not just the selected password, but also all text entered into the password field, to permit a server-side strength meter to be used. A defence-in-depth approach would attempt to minimise the risk of exposing sensitive information such as this, but it was demonstrated that this is not the case, with an adversary holding a valid CA certificate for Google's domain able to do this. This is a scenario which has been seen in the past on a number of documented occasions. This highlights the risk of relying upon a mobile device for secure computing and access to data within a decentralised network, since an application could be run on an Android device to permit access to a decentralised storage network, but the device would still be vulnerable to the issues identified here, highlighting the risks of existing centralised platforms and services upon the endpoint used to access the decentralised services.

The analysis of a range of Android encryption applications highlighted the importance of security claims being challenged; widely used Android apps claiming to encrypt user data were found to either use heavily flawed encryption permitting plaintext recovery, or to not use encryption at all. A home-made monoalphabetic substitution cipher was found in the second app listed on the Play Store for the search query "encrypt". Several of the apps also leaked the user's selected PIN credential for file "encryption" as an unsalted hash, permitting easy recovery of the original PIN entered. This highlights the importance both of well-audited software, and of not relying on the end application to provide encryption of user data. With many of the applications investigated featuring cloud synchronisation features, there is the potential for malicious service providers, or third parties able to breach these services and access user data, to retrieve plaintext data which users believe to be encrypted. This highlights a key risk of decentralised services; namely that the implementation of user data encryption must be strong enough to withstand attacks from all third parties, since user data will be exposed to others. The applications considered in this chapter did not do this, and user data would have been at risk.

Having now identified important factors in securing endpoint devices in this chapter, and ensuring the performance of access to a decentralised storage network in Chapter 4, Chapter 6 shall focus on the mapping between users and devices; for decentralised services to be usable, users must be able to interact with other people easily, discovering their public keys and identities within the network, without relying upon a centralised authority to do this, as is carried out currently.

Chapter 6

Preserving Privacy in Centralised and Decentralised Discovery

6.1 Introduction

As discussed in Section 2.6.4, a common challenge between centralised and decentralised services alike is that of discovery in a privacy-preserving manner. One particular area where this is highly visible and relevant is that of user discovery. Many modern social and communications applications encourage users to use their telephone number to find existing contacts who also use the service. This allows uses to boostrap their existing social network, and is beneficial to the service provider as it reduces user friction in beginning to use the application.

The state-of-the-art in this technique, as discussed in Section 2.6.4, is to carry out a manual intersection of telephone numbers server-side, and return the account identifiers of other service users who are listed in the user's contacts list. The major downside of this approach is that it reveals a user's social graph to the server, and permits tracking of users across services, since their telephone number is static across all services. This may be a significant privacy issue for services where users might want to control who can see that they use a service.

Despite this, there is no clear privacy preserving technique which offers sufficient protection against the server operator to prevent trivial identification of users' social graphs. The state-of-the-art in privacy preserving user discovery is that carried out by Signal (previously known as TextSecure), where partially-truncated cryptographic hashes are used to provide some level of ambiguity between users, although this is imperfect, as highlighted in Section 2.6.4.

This chapter shall contribute an exploration of the techniques available to perform privacy preserving service user discovery, define the problem, and present a solution to the problem which offers security against the service operator identifying users' social graphs in an efficient manner on the server, through the use of deliberately slow cryptographic hashing and the pairing of contacts in a novel manner. The complexity of an attack is calculated and demonstrated, and a security analysis of the solution is presented.

6.2 Privacy Preserving Service User Discovery

Despite phone numbers not being a strict secret (like that of a secret encryption key), since they must be exchanged for calls or SMS messages to be sent and received, past research has been conducted into means to preserve number privacy. Previous work has highlighted this, exploring how users can preserve the privacy of their long-term phone number, while still being contactable, and discuss the use of temporary phone numbers as a way for users to avoid having to give out their persistent and permanent phone number to untrusted third parties [210].

Given the potential for breach of privacy which could occur if a list of phone numbers are service user identities were to be disclosed from a service gather such data [211], it is therefore of clear benefit to user privacy for a central server to not be able to observe the phone numbers of users and their contacts while attempting to find other users.

Privacy preserving service user discovery is therefore a related problem to that of Cristofaro et al.'s *Private Contact Discovery* [212], and potential solutions may be sought from the field of private information retrieval. The problem differs from these, however, since the goal is for a user to determine which of their contacts also uses a service, rather than to privately carry out an intersection of their contacts with another individuals' contact list. The problem differs since under private contact discovery, both parties carrying out the intersection have access to their own contact lists. Under service user discovery, the server aggregating each user's contact data representation should not have access to the underlying contact data. To preserve user privacy, the identifier of any given user of a service (i.e. their phone number), as well as that of their contacts, should be inaccessible to both the server and other service users, although the value retrieved by a user need not be obscured from the server, provided it does not reveal user phone numbers.

A successful implementation of privacy preserving service user discovery therefore requires that;

- Phone numbers or other identifiers are not available to those who do not already know them.

- Mutual contacts should be able to determine they both use the service.

- It should be impractical for users and server operators to infer information about another party's other contacts.
- A user need only know another user's phone number (or other identifier) to use the protocol
- The server and its operators should not be able to see user phone numbers or generate social graphs

6.2.1 Approaching the Problem

Telephone numbers are designed to be relatively short for the purpose of manual entry and sharing, and comprise only a limited character-set of Arabic numbers for entry on numeric keypads, therefore making them inherently low-entropy. While various mutual contact negotiation protocols have been proposed, including high performance ones [213], these requires high entropy (non-enumerable) inputs to be used, which is not the case for telephone numbers. To illustrate this, if it is assumed that every digit from 0 to 9 is equiprobable at each position within a telephone number (to give a high estimate of entropy), a ten digit telephone number has a maximum of 33 bits of entropy, based on a per-symbol entropy of 3.32 bits, as shown in Equation 6.1, as derived from [214].

$$H(x) = -\sum_{i=1}^{n} p(x_i) log_b p(x_i) = -log_2 \frac{1}{10} = 3.32 bits/symbol \tag{6.1}$$

Entropy therefore cannot be created and introduced to a telephone number, without rendering it unusable for contact discovery, since the other party would not be aware of the extra data added to the number. However, it is possible to double the effective entropy of a telephone number, by storing paired telephone number records, combining the identifiers of the two contacts, rather than storing individual telephone numbers. If user discovery is considered as a process carried out between discrete pairings of mutually connected users, it is possible to remove any reliance upon individual phone numbers (which are relatively low entropy). This fits with a clear potential use-case, whereby two people are mutually connected (that is, both have the other in their contacts list). Both parties could therefore derive the same contact pairing.

Each user of the service would generate a list of potential pairings, based upon a combination of their phone number, and that contact's number. This process is entirely deterministic, and repeatable by the other party (who is in possession of the same two phone numbers). To avoid ambiguity, it is necessary to ensure that the pairing will always be formed in the same order.

6.2.1.1 Privacy Considerations

Despite the use of hashing of phone numbers, this still permits a service operator to carry out generation of a social graph. Social graphs generated from data gathered from mobile devices have previously been shown to reveal significant information about users' social ties, indicating there are naturally privacy considerations here. Potentially more significantly, however, since a service utilising phone numbers for discovery carries out the process on each of its users' address books, it would be possible for the server operator to build a social graph incorporating individuals who have not chosen to use the service, simply by identifying which hashed phone numbers are queried by which users.

> In order to provide the WhatsApp Service, WhatsApp will periodically access your address book or contact list on your mobile phone to locate the mobile phone numbers of other WhatsApp users ("in-network" numbers), or otherwise categorize other mobile phone numbers as "out-network" numbers, which are stored as one-way irreversibly hashed values. [215]

6.3 Novel Approach to Private Service User Discovery

A novel approach to the problem would be that of the formation of contact pairs. In this approach, one party forms a contact pair with each of their contacts, based upon the combination of both parties' phone numbers in a predictable manner. This contact pair is defined as the concatenation (with deterministic ordering) of the originating user's phone number, and that of their contact. Forming these contact pairs doubles the difficulty of attacking a hashing algorithm. This can be demonstrated by the number of possible permutations of phone numbers. By considering a situation where there is a set, x, of all possible phone numbers, from which all numbers are issued, then an attempt to carry out a brute force attack on a hashed phone number will require, on average, $\frac{x}{2}$ attempts to identify an individual number. In contrast, if contact pairs are used, an attacker can determine 2 phone numbers in $\frac{x^2}{2}$ attempts. This presents a significant increase in work; for the case of the USA where there are around 2.8 billion possible phone numbers, it would take an average of 1.4×10^9 hash calculations to determine a number. If contact pairs were used, however, such an exhaustive attack would take an average of 3.9×10^{18} attempts.

The increase in difficulty of carrying out an exhaustive search of a hashspace presents potentially significant benefit to the privacy of service users, since it is possible to double the entropy of a phone number through the use of a pairing. As a consequence, this increases the difficulty of carrying out an attack by many orders of magnitude. Indeed, and intuitively from the calculations presented above, the more potential phone numbers, the exponentially greater the difficulty in carrying out a brute force attack. To further increase the security against a brute force attack, a memory-intensive hashing function, such as scrypt, may be used to slow down and increase the cost of attacks [216].

Since the scrypt function requires a salt input, the salt input should be selected as the cryptographic hash of the concatenation of a service-specific identifier, to prevent a service-independent rainbow table being created by an adversary. The two telephone numbers then form the input to the hash, which is the contact pair, separated by a delimeter. By placing the service salt (a component known to any adversary) at the beginning of the hash, this is thus protected against hash length extension attacks against the service-specific salt. This is important to prevent two services, one whose salt was the prefix of another's, re-using their pre-image calculations by carrying out a hash length extension attack [217].

Once a client has computed a series of contact pairs, it may query the server for the scrypt outputs of these, and identify if such an output has previously been lodged with the server. If this is the case, this would indicate that the other contact in the pair uses the service, and has published that they wish to be visible to the user in question.

6.4 Background on Memory Hard Hashes

The scrypt hash function is a memory-hard, deterministic, key derivation algorithm, designed to be resistant against brute-force attacks from dedicated hardware, as well as general purpose computers [216]. It features tunable difficulty parameters, allowing the required memory and processing power to be adjusted to meet the needs of the implementation. In this section, it is demonstrated that scrypt is a memory-hard hash, which is suitably difficult to parallelise, and that its parameters offer a wide degree of tuning, allowing RAM and CPU requirements to be tailored. This is significant in ensuring that the solution remains practical for mobile phones to implement, yet remains difficult to attack using large-scale computing resources.

The scrypt function takes three parameters related to performance; which shall be referred to as N, r and p, to maintain consistent notation with previous literature [216]. N is the CPU/RAM cost parameter, r is a memory multiplication factor, and p is a parallelisation factor. Increasing any of these parameters results in a slower calculation of the resulting output hash, with a linear increase in processing time, as each parameter is increased. Of note is that the cost parameter, N, gives an exponential rise in processing time, since the scrypt algorithm calculates the cost factor as 2^N.

This scalability of performance makes an scrypt-like algorithm ideal for tunable performance, to ensure parameters offer sufficient resistance to attack, while offering acceptable performance on mobile phones.

6.4.1 Performance on Mobile Phones

Since the proposed contact discovery protocol places significant computational load onto the client device (and indeed this provides the basis of the security of the proposal), achieving suitable performance on mobile devices is critical to the viability of this solution. For the purpose of consistency, all tests were repeated 20 times, and the average of these runs is presented. Additionally, each Android device was tested while plugged into a computer, with the screen enabled throughout the test, to ensure that performance was not limited by the CPU governor reducing the processor's clock speed in sleep.

These results were carried out using a native C implementation of the scrypt algorithm, invoked using JNI (Java Native Interface) from a standard Android application, and are thus achievable by an application implementing this protocol. For these measurements, only a single core was used for the scrypt process. No other applications were running on the device at time of running the test, and the devices were all disconnected from their respective wireless or mobile networks (to prevent incoming push messages from triggering the handling of incoming messages).

For each of the three scrypt parameters, the value was varied across a range of values, commensurate with the bounds given in previous work, as shown in Equations 6.2 and 6.3 [218]. This was to allow for the identification of suitable parameters offering sufficiently high performance on mobile devices, yet offering resistance to large-scale attacks from an attacker.

$$1 < 2^N < 2^{128 \times \frac{r}{8}} \tag{6.2}$$

$$0 < p \leq \frac{2^{32} - 1 \times 32}{128 \times r} \tag{6.3}$$

By rearranging Equation 6.3, Equation 6.4 can be derived, which constrains both p and r parameters of the scrypt function. This is as expected from the definition of scrypt, where a unit increase in any parameter results in a unit increase in the number

Table 6.1
Android device performance and chipset comparison

Device	Chipset	Release	Secs/Unit Work
Motorola Defy	TI OMAP 3610	10/2010	4.20×10^{-6}
Samsung Galaxy S2	Exynos 4212	04/2011	4.85×10^{-6}
Samsung Nexus 10	Exynos 5250	11/2012	1.83×10^{-6}
Samsung Galaxy Note 2	Exynos 4412	11/2012	2.56×10^{-6}
Sony Xperia T	Qualcomm MSM8260A	09/2012	2.58×10^{-6}
Sony Xperia Z	Qualcomm APQ8064	02/2013	2.67×10^{-6}
Sony Xperia Z Ultra	Qualcomm MSM8974	06/2013	1.75×10^{-6}
Asus Nexus 7 (2013)	Qualcomm APQ8064	07/2013	2.68×10^{-6}
Oppo N1	Qualcomm APQ8064	10/2013	2.51×10^{-6}
Sony Xperia Z2	Qualcomm MSM8974AB	04/2014	2.00×10^{-6}
Oppo Find 7	Qualcomm MSM8974AC	05/2014	1.60×10^{-6}

of loop cycles executed.

$$0 < p \times r < 2^{30} \tag{6.4}$$

By altering one scrypt parameter at a time, while holding the other two constant, it was determined that each parameter had a linear effect on the execution duration of the scrypt function. As such, by monitoring the execution duration of 72 different scrypt tasks (each repeated 20 times for accuracy), for all values in the ranges of $2^{10} \leq 2^N \leq 2^{13}$, $10 \leq R \leq 15$, $1 \leq P \leq 3$. Since the performance of the algorithm was previously found to be linear in relation to each parameter, a work factor was also introduced, and calculated for each problem, as shown in Equation 6.5 where w refers to the work factor for a given set of parameters.

$$w = 2^N \times R \times P \tag{6.5}$$

Table 6.1 shows the calculated performance per unit work factor, calculated for a variety of Android devices released between 2010 and 2014. This selection was made to include devices including different chipsets, of varying ages.

Combining these measured performance parameters with Equation 6.5, is possible to predict the time taken for a given mobile device to carry out a given scrypt calculation, and to thus establish the necessary parameters of the scrypt function, based on the desired time taken on a particular model of mobile phone.

6.4.2 Parallelisation of scrypt

The scrypt algorithm, being memory-intensive and inherently difficult to parallelise, is designed specifically to be difficult to scale in a cost-effective manner, including through the use of GPUs [219]. Similarly, the scrypt algorithm is designed to resist attacks by custom hardware, through the costs involved in the creation of such a device [216]. In order to demonstrate this is true, and establish the viability of a larger-scale attack, the performance and throughput of the scrypt algorithm was tested against a set of parameters, $N = 2^{17}, R = 5, P = 1$, which equates to an scrypt work factor, per the earlier definition in Equation 6.5 of 655360.

An example attack was implemented using various numbers of threads, on a dual-CPU Intel Xeon E5-2620 server, with 2.0 GHz clock speed. For the purpose of this experiment, the number of threads used to calculate 120 scrypt hashes was varied. The number 120 was selected, since it has factors of many common integer numbers of cores (allowing each CPU thread to attack its own integer number of hashes), therefore offering a generous estimate of attack performance. Since the CPUs in use supported Intel's hyperthreading technology, and thus presented 24 virtual CPU cores, rather than the 12 physical CPU cores, this made it possible to verify if an scrypt attack's performance would be improved.

Figure 6.1 shows that for up to 12 threads, the total computing time (as a sum of all tasks executed) was approximately constant - this indicated that the scrypt algorithm could be effectively parallelised for physical cores. Above 12 threads, however, the total execution time increased linearly, indicating that hyperthreaded cores were not offering any performance advantage. This linear increase in execution time continued, with 120 parallel threads requiring 9× more combined processing time than was required for up to 12 threads. This indicates that the scrypt algorithm requires (and was bounded in performance by) physical CPU cores, rather than hyperthreaded virtual cores.

Figure 6.1 also indicates that RAM usage increased linearly with the number of parallel threads executed. Therefore, even on a system with a large number of CPUs, attack performance will be constrained by RAM bandwidth and available quantity. This confirms the assertions made by Percival, as to the difficulty and cost of carrying out a large-scale attack against scrypt [216].

Having determined above that parallel CPU-based attacks against scrypt scale linearly, up to the number of physical CPUs available, the performance of a single CPU core from the server was determined, by carrying out the same benchmarking process as carried out on the mobile phones (per Section 6.4.1). A single Xeon E5-2620 CPU core required 6.73×10^{-7} seconds per unit of scrypt work (per Equation 6.5), which was 2.3× faster than that of the current-generation mobile phone tested, the Oppo Find 7,

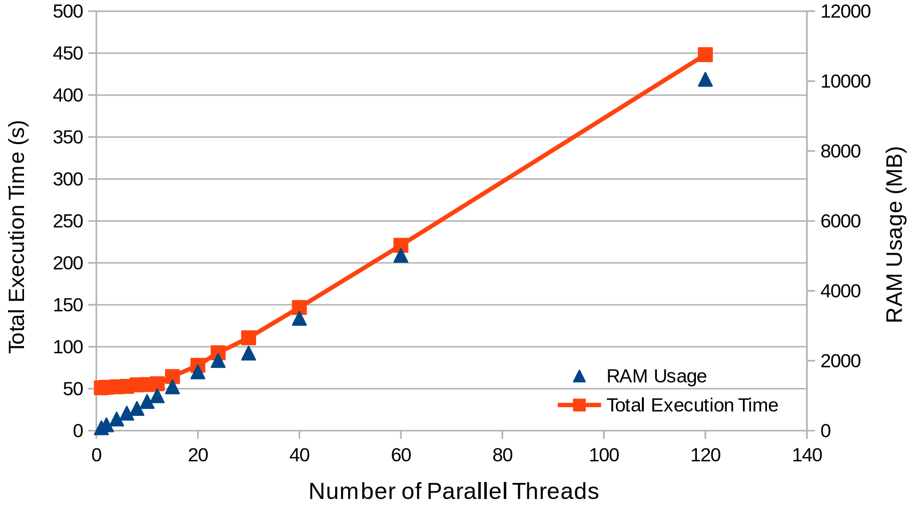


Figure 6.1
Time of execution of parallel scrypt tasks, and RAM usage, simulating an attack

with an MSM8974AC CPU.

6.5 Implementation Description and Performance

To demonstrate the practicality of this proposal in an actual implementation, a client application was created for the Android operating system. This section explores design considerations of the solution, the techniques to be implemented on the client, and the overall performance of the solution, as implemented, as well as the process used to select the scrypt parameters.

6.5.1 Client Application

The client application was an Android application, written in Java, invoking scrypt via a JNI (Java Native Interface) library. This implementation allowed for development of the Android application and user interface with the standard Java-based API, while gaining the performance benefits of a natively-compiled C library for the intensive scrypt operations. This also allowed for the reference implementation of scrypt to be used, rather than a new implementation being created. The Java application was tested against the scrypt test vectors, to verify its operation was correct [216].

This client application carried out service user discovery using the following process:

- Identify the user's phone number from their handset
- Access the user's contacts, and obtain a list of phone numbers

- For each phone number:
  - Compute the result of $scrypt(number1 + ", " + number2)$
  - Store this output as the request key
  - Take the cryptographic hash of this request key
  - Query the server for the value held under this hash
  - If a value is returned, decrypt it with the output of scrypt (as described below)
  - Continue to next phone number

For each phone number in the contacts list, a pair is created, by combining the two telephone numbers together. For a user with n contacts, there will therefore be n pairings calculated by the client application. For the purpose of efficiency, the results of scrypt on each pairing can be cached locally on the handset, meaning the scrypt calculation need only be carried out once on the mobile device.

Upon querying the server, if no existing record was found, the client then creates such a record. To ensure that the process of querying records is repeatable (such that either party will make the same request), it is necessary to deterministically select the order in which both contacts are combined. By placing the lower (numerically) telephone number first, after removal of any non-numerical symbols such as the "+" international prefix, both clients will always generate and seek the same order of pairing of phone numbers.

The process of hashing the output of the scrypt function is used, to allow the client to decrypt a message uploaded by the party making the contact pair record, without the server ever being made aware of the raw output of the scrypt function. This message should not be presumed secure against all attacks (any party knowing both telephone numbers may retrieve it), but it allows for two parties to exchange a service-based identifier such as a username, without the server becoming aware of either party's identifier on the service. This process is entirely optional, and no value need be stored on the service if this is not desired; it is merely possible to be carried out if desired.

For the server to decrypt this record, it would require knowledge of the output of the scrypt function for the same inputs as used to encrypt the record, meaning that it would be necessary for the server to know the phone numbers of both parties. Determining these would, from the perspective of the server, effectively require the server to carry out a full pre-image attack across all potential phone numbers, therefore resulting in considerable resources being required. This scenario is considered in more detail in Section 6.6.

6.5.2 Server Application

The server-side implementation is designed to be similar to that of the Signal server implementation, in that a simple API can be provided for client queries. As this proposal for the user-discovery portion of a service can be built around key-value storage, the Redis key-value pair storage backend was used for performance. During development, the API was also tested with the Postgres database. Users present the server with a request for information regarding a given contact pair (where this request is the result of the computationally intensive scrypt process), and receive either an error to signify such a pair not existing, or a response containing the value associated with that pairing (a non-error return). With only two new API functions required, this server application is easily implemented on top of an existing communications system.

6.5.3 Parameter Selection

In order to preserve user privacy against attackers with significant computing resources, it is necessary to balance the scrypt parameters, such that exhaustive attacks are infeasible, yet users can generate their own contact pairs in a reasonable time. Client devices should only need to generate a contact pair once, as it can be stored locally for future queries against the network, thus reducing computing power used after the initial preparation. Having determined in Section 6.4.2 that the scrypt function on the tested server CPU was around only 2.3× faster than the fastest mobile phone available today, the desired performance on a current-generation mobile phone can be balanced with that of the attacker.

To determine suitable example scrypt parameters, a scenario should be considered, with a user of a US or UK-based mobile user, using a mobile handset at the end of its typical lifespan of 2 years [220], to identify worst-case performance for a typical user. In this case, the 2-year old Sony Xperia Z is considered as the reference handset. It required 2.67×10^{-6} seconds per unit of scrypt work), as determined in Section 6.4.1.

In order to ensure that the process is suitably responsive, it should be ensured that the process of contact discovery for one contact takes less than a second, such that the user can see clear activity taking place. For a user with 150 contacts, which is Dunbar's number, the maximum number of stable social connections a person may maintain [221], the process should therefore take less than 150 seconds to complete (in the background). As demonstrated previously, performance of scrypt is scalable by physical cores, so this could be achieved by parallelising the process of contact discovery. For the purpose of ensuring a responsive user experience on other applications on the phone, only 1 CPU core will be considered usable be used for contact pair generation.

Based on this limit of 1 second per contact, as a worst-case scenario for a 2-year old handset, it is possible for an scrypt work factor of 3.74×10^5 to be used, per Table 6.1 in Section 6.4.1. To determine suitable parameters, Equation 6.5 is used, solving for the highest possible value of N, while keeping $P = 1$ (to maximise memory usage), adjusting R as necessary to obtain the closest possible solution. Therefore, suitable parameters for the scrypt function, under these constraints, would be $N = 2^{17}, R = 3, P = 1$, yielding a work factor of 3.93×10^5.

Based on these parameters and the modelled performance of both the server and mobile phones, it was predicted that that each scrypt hash should take 1.05 seconds to calculate on the phone, and 0.26 seconds on a server CPU core. This was confirmed experimentally to be accurate, and matched with the predictions made from the work factor calculations detailed in Section 6.4.1.

6.6 Potential Attacks

In this section, potential means through which an adversary may attack this system are considered. The performance of this proposed solution under these scenarios is considered.

In a user-targeted attack, a user (of known phone number) is singled out by an adversary, and as much information as possible about this user is sought from the network. In an untargeted attack, an adversary wishes to gain as much information as possible about as many users as possible, such as their telephone numbers or social relationships. In order to evaluate the security of this solution, and the extent to which it provides security, the requirements of a privacy preserving solution must be considered, per Section 6.2. The principal goal of this solution is to prevent a party from evaluating the full range of possible telephone numbers, thus identifying which users are present on the service and their social graphs, which is possible to be carried out on current state-of-the-art implementations, as discussed in Section 2.6.4.

In a social graph discovery attack, the attacker is assumed to wish to evaluate and explore the social graph of a user, for the purpose of identifying their contacts, and the contacts of those contacts. Such an attack can be continued for a number of iterations, in order to find users located more distantly through the social graph. An active and effective attack can also use the information gained through exploring the social graph to speed up attacks within future branches of the graph.

These analyses presume that the service provider is actively colluding with an adversary, such that attacks are not constrained by network latency or server-side rate limiting. As such, a real-world attack against a non-colluding service would take significantly longer, without direct access to make unlimited queries to the service database.

Within this solution, the server has access to the full database of user key-value pair records. The key of each record is the cryptographic hash of the output of the scrypt algorithm for a given pair of telephone numbers. The value of this data is an encrypted representation of the uploading user's service identifier, encrypted by the direct output of the scrypt function. Since the server does not receive the raw output of the scrypt function (only its hash), then without a suitable pre-image attack against the cryptographic hash, the server cannot decrypt the value without knowing the original input to the contact pair, and carrying out the scrypt operation itself to derive the key.

6.6.1 Untargeted Attacks

In an untargeted attack, it is necessary for an adversary to generate a set of telephone numbers, within which they wish to carry out their attack. As the size of this set increases, the computation necessary to carry out the attack increases quadratically, due to the pairing of contacts (thus doubling the entropy of each record), as explored in Section 6.3. Since this attack is based around the goal of identifying user telephone numbers, it should be assumed that the adversary does not have access to a list of valid telephone numbers. If an adversary were to have access to a list of valid telephone numbers, this attack would be unnecessary, since the adversary already has such a list, and would simply be wasting computing resources. While it would be possible to use such a list to carry out mappings of a user's social relationships, this would be a series of targeted attacks, since the adversary has already acquired a target user's telephone number. These are considered in Section 6.6.2.

To identify all possible service users with US-based telephone numbers (a set of 2.8 billion, based on around 10 million possible numbers in around 280 area codes), it would be necessary to generate $(2.8 \times 10^9)^2 = 7.8 \times 10^{18}$ total contact pairs. Each would then need to be evaluated through the scrypt function. Since every service uses a different service identifier within the scrypt salt, this process must be carried out for every service an adversary wishes to probe. Per the parameters selected in Section 6.5.3, it was established that the scrypt process recommended by this solution would take approximately 1 second on a 1-year old mobile phone, and around 0.26 seconds on one core of our server.

If an adversary wished to evaluate the entire set of contact pairings in under y years, it would be necessary for them to run N parallel attack instances, where N is established from Equation 6.6, P is the number of contact pairs to be evaluated, s is the work-performance rate of the CPU (6.73×10^{-7} seconds per unit scrypt work for the server considered in this work), w is the number of units work required per hash (3.93×10^5), and 3.154×10^7 is a constant, the number of seconds in a year.

$$N = \frac{P \times w \times s}{y \times 3.154 \times 10^7} \tag{6.6}$$

In this case, to carry out an attack in under 5 years, 1.31×10^{10} (13.1 billion) of our server CPU cores would be necessary, in order to evaluate all permutations of the 2.8 billion possible telephone numbers. Note that it is not possible to short-cut this process, while still evaluating all telephone numbers, as an adversary cannot establish if a contact pair reveals duplicates of already-discovered telephone numbers, unless they exhaustively search the full range of contact pairs. One would assume there are considerably easier and more cost-efficient means of identifying lists of user telephone numbers, and that this level of computational power is not viable to use to establish social graphs of users. Since these calculations are based on evaluating only telephone numbers within the United States, the introduction of more potential phone numbers will exponentially increase the number of pairings to evaluate.

With a thermal design power (TDP) of 95 watts per CPU [1], with 6 hardware cores, 13.1 billion CPU cores would require on the order of magnitude of 2×10^{11} watts of power. This would result in energy consumption over the 5 years of 9×10^{15} Watt-hours. By way of comparison, the UK's annual electricity consumption for 2014 was 3.09×10^{14} Watt-hours.

Since this attack duration is based on parameters selected to offer reasonable performance on a 2-year old mobile phone (per Section 6.5.3), it would be possible to increase these parameters to account for future advances in CPU technology, depending upon the desired level of security.

While it is not possible to accurately determine the number of possible telephone numbers in the entire world, since this would require an understanding of the (often not publicly undocumented) numbering scheme of every individual country, if it was assumed that a country had, as a conservative estimate, 8 unique digits per telephone number (noting that UK and US numbers contain more, with 10 such digits), there would be 1×10^9 possible telephone numbers per country to evaluate. With 196 countries, this would lead to a total of 1.96×10^{12} possible telephone numbers, and thus 3.84×10^{24} contact pairs. To evaluate these, per the agreed scrypt parameters, again within 5 years, would require the dedicated use of 6.44×10^{15} server CPU cores, based on the CPU used for this work. As discussed in Section 6.4.2, this attack is not economically scalable through the use of dedicated hardware or GPUs, on account of the

[1] http://ark.intel.com/products/64594

memory-hard nature of the scrypt algorithm.

6.6.2 Targeted Attacks

In a targeted attack, the phone number of the target user must be known. Therefore, an adversary may wish to identify the contacts of this user, through either an exhaustive, or targeted, search. When one half of the contact pair is known (in this case, the target user), the difficulty of carrying out a targeted attack is reduced to that of evaluating the potential contact telephone numbers (rather than all permutations of contact pairs).

One specific attack which can be carried out under our solution is a contact confirmation attack, whereby an adversary is able to confirm if two given users are in contact with each other. For this to be viable, it is necessary for the adversary to know the phone numbers of both users. The solution proposed here would allow the third party to confirm that at least one of the two users has registered with the service in question, and that they have the other user in question added as a contact. The adversary cannot prove reciprocity of contact, or that contact has taken place. Similar information can already be gathered by the server operator of state-of-the-art services, and indeed access to an individual's contact list would reveal this information. Indeed, it is also not possible to determine at this point which contact of the pair is a user of the service — both parties would upload the same contact pair record, therefore it is not possible at this stage to determine which of the two contacts uses the service.

It is worth noting, however, that this attack is not scalable, and requires the adversary to have specifically identified two individuals to verify a relationship between, having already established their telephone numbers. For such an attack to be viable, an adversary must isolate a single user's potential contacts to a relatively small number (much smaller than the full range of telephone numbers). Otherwise, this attack becomes a generic targeted attack aimed at discovering an individual's contacts. Having been able to establish this subset of potential contacts themselves, however, the adversary would likely be simply confirming what they already knew - they would have had to already establish the presence of relationships between users to make this attack meaningful.

It is worth noting that while discovery of an individual targeted user's contacts remains computationally feasible, it remains (in absolute terms) relatively computationally intensive. As such, the attack is not scalable to significant numbers of users, and tends towards a regular targeted attack at that point. Even then, an individual of interest is already making available their telephone number to their legitimate contacts, and social engineering of these contacts would likely yield more rapid (and cost effective) results.

The difficulty of evaluating an individual user's contacts on the service may be calculated through the process described in Section 6.6.1 — in the case of a targeted attack, seeking only USA phone numbers, (2.8×10^9) scrypt operations would need to be carried out to evaluate all possible contacts of the user in question. Using Equation 6.6, with the parameters selected in Section 6.5.3 ($P = 2.8 \times 10^9$, $s = 6.73 \times 10^{-7}$, $w = 3.93 \times 10^5$), 23 Xeon E5-2620 CPU cores would need to be dedicated to evaluating one user's contacts with phone numbers in the USA, to complete the process within in a single year. To evaluate contacts worldwide, based upon the numbers calculated in Section 6.6.1, 16,430 Xeon E5-2620 CPU cores would be required to evaluate one user's contacts list in a single year.

While these figures do indicate it would be practical to carry out such an attack against an individual's contacts list, the barrier to carrying this out is significantly increased in this solution — it is not possible for the service operator to collude to provide the information, as it is not available to them. It is also not possible to form a social graph from access requests, since records are pairing-based, and access records will therefore only ever be recorded from one party. Similarly, this attack requires a linear increase in computing resources to scale; carrying it out against another individual will require the same level of computing power as the first would require. Finally, it is also possible that an adversary would simply be put off from carrying out an attack of this scale — there could be better returns available from using the hardware to mine Bitcoins, rather than attempting to find out the phone numbers of an individual's contacts.

6.6.3 Social Graph Discovery Attack

In a social graph discovery attack, the objective of the adversary is to attempt to explore a user's social graph. If there is a target user with a known telephone number, they can be used at the centre of the attack. Such an attack will first seek to identify this user's contacts, which is effectively a targeted attack. This would require (for a USA-only scenario as discussed above) 2.8×10^9 operations to be carried out. This would identify the user's first-order (directly known) contacts. From this point, a more rapid attack can be carried out to intersect each of the user's known contacts with each other; if it is assumed that the target user had 150 contacts (per Dunbar's number discussed previously), this would result in a further 22,350 operations to be carried out (each contact being checked against each other).

As was established in Section 6.6.2, the untargeted component of this attack would require 23 of the reference CPU cores to evaluate one user's contacts against all possible US telephone numbers within a year. Under this attack mode, it would also highlight any links between mutual contacts of the target. To continue to further propagate the attack to second order contacts or beyond, it would be necessary to carry out $150 \times 2.8 \times 10^9$ look-up operations; one for each of the contacts of the first user. This would, on average, yield 150 contacts for each of those contacts. At this point, there would likely be some duplicate contacts, but in a worst-case scenario of no interconnection between second order contacts, the links between second order contacts could be established with $(150^2 \times (150^2 - 1) = 5.1 \times 10^8$ further operations (150 contacts for each of the original target's contacts, each checked against each other).

This attack therefore presents an opportunity to explore the social graph of an individual. While it initially begins as a targeted attack around an individual's contacts, these can then be used to carry out more targeted attacks, if information about the connections between such users is desired. To explore beyond the first order of a user's contacts, however, will require more targeted attacks to be carried out, and each of these requires 23 core-years of the reference server CPU to carry out. Therefore, while the attack is initially feasible to ascertain the first-order social graph of a contact, going beyond this to exhaustively search the "contacts of contacts" requires a targeted attack to be carried out for each user. At this point, the second order graph could be evaluated in 4.3 core-years of work on the reference CPU, testing each "contact of a contact" against each other, to establish connections.

If particularly well-connected nodes are identified amongst the social graph, these could become the subjects of their own targeted attack, to exhaustively discover all of their own contacts, and potentially reduce the need to carry out more exhaustive targeted attacks, albeit with the trade-off of not necessarily identifying every user in the social graph.

6.6.4 False Friendship Attacks

As discussed in previous work, when considering contact discovery protocols, it is naturally desirable for service users to be able to prevent unwanted invites from causing annoyance or irritation [212]. In this implementation of privacy preserving service user discovery, this is addressed by requiring that all contacts are reciprocated. Since a service user will only attempt discovery of contacts they know, it would be possible for a user to publish a contact pair for a large number of users. In reality, server-side rate limiting, and the computational work necessary to produce each contact pair would most likely discourage this behaviour from most users. Even if this were to happen, only a user attempting to locate the first user would ever find out about this request, as without knowing the requesting telephone number they will never discover the contact request.

As such, this solution does not allow for a user to receive unsolicited contact requests. Since user will only discover contact pairing requests from users whose telephone number they know, there is no direct harm from the generation and publication of false contact pairings. Indeed, these may also provide an element of plausible deniability, or at least distraction for an adversary. The introduction of decoy contacts is discussed in Section 6.8.

6.7 Comparison and Evaluation

In order to evaluate the performance and security of this solution, compared to the state-of-the-art, we compare an implementation of this privacy preserving user discovery system to version 0.31 of the open-source TextSecure server [2]. The privacy risks posed by the state-of-the-art are compared to the threats identified discussed in Section 6.6, and the overall performance and ease of implementation is compared.

6.7.1 Comparison with TextSecure Server

The TextSecure service operates around a semi-trusted server security model, where the content of messages between users is encrypted, but the user discovery protocol is visible to the service operator. While operating a local TextSecure server instance, it was possible to easily obtain a list of the telephone numbers of all service users, by running a single SQL query on the database, namely `SELECT number from accounts;`. It should be noted that this was not due to the contact discovery process; rather it was due to the inherent need for a TextSecure server to validate user phone numbers. Nonetheless, hashes of contacts were also made available to the server, and could be retrieved as the server operator. In the event of the contact discovery server being compromised by an adversary (or an untrustworthy entity operating it), all user telephone numbers may be exposed.

A social graph of links between service users could also be formed by the operator of a TextSecure server, or indeed anyone gaining access to it, on account of its contact discovery process. The HTTP request from client to server supplies a set of contact tokens to query for existence on the service, each being the unpadded base-64 representation of a truncated SHA1 hash of the contact's telephone number, as discussed in Section 2.6.4. Since the SHA1 hashing algorithm is highly vulnerable to a brute-force attack across the full phone number space, and the service operator already has full access to the list of all telephone numbers on the service, it would therefore be relatively straightforward for the server operator to create a mapping between telephone numbers and the corresponding contact token received. Indeed, this is how the contact discovery process functions internally, since the TextSecure server maps the incoming contact tokens to existing accounts (which are identified by their telephone number).

The TextSecure contact discovery solution therefore reveals to the service operator

[2] https://github.com/WhisperSystems/TextSecure-Server

both the identity of every user, and their full social graph of contacts. It is also possible for the service operator to identify if a given telephone number is a user of the service (per the list of all users available in the database). It must be noted however, that this conclusion is based solely upon technical analysis of technical possibility — there is no reason to suggest the operators of such a service would carry this out; rather this highlights what a malicious service provider or party gaining access to the servers could do, or could be compelled to do.

6.7.2 Contribution Overview

In contrast, the contribution of a privacy preserving user discovery solution offers protection against this kind of data collection by a service operator. Indeed, the above limitations of the TextSecure server implementation are the equivalent of untargeted attacks (since it is viable to carry them out against every service user). In contrast, as justified in Section 6.6.1, it is impractical for a service operator to identify the telephone numbers of all service users. Under the solution described in this chapter, it is also considerably more difficult for a service operator to identify if a given user makes use of the service, since this forms a targeted attack, as per Section 6.6.2, as well as the knowledge of one of their contacts, which requires significant resources, proving costly to use. From the perspective of an adversary with access to the server, this solution offers users significant privacy gains, compared to the state-of-the-art, and significantly increases the complexity of an attack, as justified in Sections 6.6.1 and 6.6.2.

The state-of-the-art protocol is designed to allow any user to determine if any other person makes use of the service, simply by searching for their phone number. With this solution, a third party cannot easily determine if an arbitrary telephone number makes use of the service, unless they know the phone number of a user which is a mutual contact of the target. In this case, it is possible to confirm that either of the two parties uses the service, and that one of the two parties has declared a relationship with that telephone number.

Indeed, while this may initially appear to pose a risk to users, in that a knowledgeable attacker could potentially establish information about their social pairings, surprisingly little information can be ascertained from the publication of a contact pair — since either party may publish it, it can only be determined that:

- One of the two parties uses the service
- One of the two parties has published a contact pairing record for the other

The identity of the party publishing the contact pairing cannot be determined by an adversary in these circumstances, even having identified the phone numbers forming the pairing, since the ordering is deterministic (as described in Section 6.5.1). Likewise, while it is possible to tell that one party published such a contact pairing record, the link may not be mutual, or this party may have published a decoy record, as discussed in Section 6.8. The processing necessary to carry out this attack also increases linearly with the number of potential contacts to attempt to match against. Therefore, while it is possible for an adversary to make suppositions as to contact relationships between entities, they would require a small list of potential telephone numbers to identify links between, indicating they likely already were only able to establish information they already had. In any case, existing social networks already offer significant quantities of publicly available information as to contacts, and therefore reduce the risk of being able to identify potential contact pairings. Indeed, Farahbakhsh et al. [222] found that a user friend list was the most common information for users to make available publicly on Facebook, with 63% of users allowing public access to their friends list). While it is possible for a determined adversary to evaluate all potential contacts for an individual user, this attack is highly time consuming, and requires as many scrypt operations as contacts as the adversary wished to attempt. These may well also yield a number of false hits, if the user made use of decoy records, per Section 6.8.

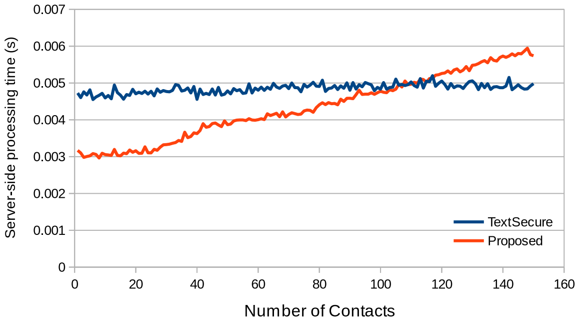
The solution presented here also inherently features protection against unwanted communications using the service, since it requires that users have mutually elected to communicate with each other, for discovery to occur. While this naturally provides a small reduction in flexibility of the service, it also ensures mutual consent is established before users are able to see each other on a service — this is inherently necessary in order to prevent a third party from determining if a given individual uses a particular service, but also offers improved privacy, since it would prevent another user from determining someone uses a given service, unless they choose to make that information available to them.

6.7.3 Performance Comparison

Despite the increase in client-side computation necessary in order to generate a contact pairing, the server-side performance of this solution should be comparable to that of other contact discovery implementations, as there is no increase in computing required on the server. To verify and prove this, the time required to carry out a server-side query to find 150 potential existing contacts was carried out against both an offline TextSecure server instance, and an implementation of this solution. Both TextSecure and this solution's implementation used the Redis key-value store for the handling of contact discovery queries, to ensure the backend technology was not skewing results.

As shown in Figure 6.2, this solution offered comparable server-side performance

Figure 6.2
Server-Side Performance Comparison with TextSecure



to TextSecure, while tested on a local system. Results shown were averaged over 20 executions of each set of tests. Tests included upto 150 contacts, on account of Dunbar's number [221]. The performance of this proposed solution was slightly faster for smaller contact lists, and TextSecure was slightly faster for larger contact lists (above around 110 contacts). Nonetheless, with each query taking around 3 to 6 milliseconds, the server-side performance offers no perceivable difference in performance — the latencies involved in connecting to a remote system significantly exceed these, especially over mobile connections [223].

6.8 Attack Mitigations

Since this proposal requires mutual recognition of a relationship for users to find contacts, it is possible for users to dilute the information gained by a highly determined adversary launching a targeted attack, by generating *decoy* contact pairs. These contact pairs would be generated locally by the client device, and uploaded to the discovery service, just like any other contact pair. In the event of a user being targeted, the adversary would discover both the user's legitimate contacts, as well as the decoy contacts, provided the decoy contacts were within the range of telephone numbers being evaluated. Decoy contacts do not pose any risk or significance to other users, since contact pairs can only be discovered by a party which seeks to find them. As such, client devices can add decoy contact pairs to the server (albeit at the computational expense of generating these contact pairs), meaning that a search will uncover both legitimate and false contacts, making it more difficult to generate a social graph of that user's contacts.

Since the privacy of user phone numbers is protected in this solution as a result of the expense of parallelising and rapidly computing scrypt hashes, it is necessary to consider the privacy protections in place, in light of the inevitable computational advances of the future, as a result of Moore's Law. Firstly, while this solution is presented and justified using scrypt, it should be possible to upgrade the security parameters of the hash function used, as well as to alter the function used, in future versions of the protocol. As such, the server API access URL should be versioned, allowing for deprecation (and removal of the legacy database from the server following deprecation).

A version upgrade of the protocol would permit upgraded scrypt work parameters (N, r and p) to be selected, such that the computation required to generate a contact pair increases. The influence of these parameters was discussed in greater detail in Section 6.5.3.

If the server can be trusted to act honestly (which is not assumed in the design of this solution), clients could also elect to request the removal of their published contact pairs following a successful pairing being established. This would mitigate the threat to their privacy, as the contact pairs would no longer be present or accessible to external adversaries. There are considerations as to potential vandalism, but this would be of the form of targeted attacks, and it would be straightforward for client implementations to automatically re-publish any missing contact pairings they have not had reciprocated, resulting in communication with the other party. This would mitigate the concern of random vandalism or removal of pairings. Nonetheless, this should not be considered as an aspect of the protocol's security, rather as a feature which may be used to hide the disclosure of contact pairings from other users of the service attempting to carry out targeted or untargeted attacks.

6.9 Conclusions

A novel approach to the preservation of social graph privacy against a service operator has been contributed in this chapter. This technique advances the state-of-the-art by permitting users to identify if any of their existing contacts use a service, without exposing their social graph, or telephone number, to the service provider. The solution is also design to operate in a decentralised service environment, where there is no centralised discovery server. A security analysis of the solution is carried out to demonstrate the complexity of carrying out both targeted and untargeted attacks against the solution, with techniques proposed to mitigate the risks of brute force targeted evaluations of a single user's contacts, through the injection of false positive records, which will present no adverse effect to the contact identification system. A social graph discovery attack is also presented, which does reduce the security properties offered, where an attacker is interested in identifying the contacts around a particular user, and the difficulty of this attack is equivalent to the difficulty of a regular targeted attack. This attack could be scaled up by a determined adversary to then repeat the process for well-connected nodes within the target's contacts, and carry out a targeted attack there. This process may increase the ease with which an adversary may recover an individual's social graph, although to achieve full coverage of the social graph, will require recursive targeted attacks to be carried out, raising the computation necessary in such instances.

The contributed approach makes it possible for two parties to identify that they both use a given service, without disclosing their telephone number, or any trivial derivative thereof (such as a cryptographic hash of it). By incorporating a service-specific salt, users cannot be tracked between services, and the use of the slow scrypt key derivation function across contact pairs prevents trivial evaluation of all possible contacts, since mutual contact pairs must be established in order to identify a connection between a pair of users.

Having now ascertained that a decentralised network can be used to facilitate secure user discovery, while preserving privacy and avoiding the introduction of a centralised service for this purpose, Chapter 7 will conclude the contribution chapters of this thesis by investigating how identity and authentication should be handled in a decentralised network, and how to permit users to protect their identity within a decentralised network, particularly in light of the vulnerabilities of endpoint software as identified in Chapter 5. The next chapter will therefore explore how a user can hold multiple secure identities, without requiring a complex process or multiple credentials, and ensuring that these long-term keys remain secure, even if a user's system is compromised, given that these identity keys are all that is needed for a third party to gain access to a user's personal data stored on the platform.

Chapter 7

Identity and Authentication

The concepts of identity and authentication are fundamentally inter-linked. This chapter explores these two concepts, and their importance in a decentralised storage network, including an overview of available authentication and identification technologies, and their applicability to decentralised storage. It then highlights the contributions made by this thesis towards secure identity management and user authentication, in the form of a design and implementation of a secure smartcard-based solution for the management of an unlimited number user identities in a decentralised network, without any storage constraints.

The work presented in this chapter has been published in [A15].

7.1 Introduction to Identity and Authentication

Identity, as defined by the Oxford English Dictionary, is "a set of characteristics or a description that distinguishes a person or thing from others" and "the sameness of a person or thing at all times or in all circumstances" [224]. This can be condensed to a requirement for uniqueness and consistency; that every person should have an individual identity, and each of these should be unique with respect to all others.

In the context of a traditional individual multi-user computer system, the concept of an account is used to represent identity. Every user account is unique with respect to others available, as an account must contains at least one distinct field, such as username or user ID. Within the system in question, an individual user can be identified based on their username or user ID, in an unambiguous and repeatable fashion. On each occasion an individual uses the system, they will be recognised as the same user, and be presented with the same files.

7.1.1 Relation to Secure Systems

While a user of a system's identity ensures they are able to gain consistent access to the system in question, this does not provide any security against other users presenting themselves as this user; knowledge of a user account's unique identifier, such as username or user ID, would allow an imposter to present themselves as a user. To prevent this, authentication is used in combination with an identity, such that a user is required to authenticate that they are identified as the legitimate person to whom the account belongs.

This process of authentication is the process of validating that a person presenting an identity is authorised to use it, by being the holder of it. Traditionally, authentication takes place through the use of a per-identity secret, whereby an entity may prove their identity through the knowledge of a secret. In the process of weak authentication, knowledge of this secret is proven through its disclosure to a verifier, which is typically the computer system being used. The verifier compares the secret disclosed by the user to a stored record of a representation of this secret, and can ascertain if the provided secret is correct.

This process of weak authentication establishes that, under the assumption that the secret for an identity is not disclosed or revealed to any party other than the owner of the identity or the verifier, the individual claiming to use the identity is the same individual as previously, and that they are permitted to use the identity. The assumption of secrecy of this authentication secret, typically a password, is however not necessarily a suitable assumption to make. It also fundamentally requires the existence of an external entity, trusted to handle and validate secrets without revealing them to others. Similarly, it requires that users take adequate precautions to prevent their authentication secrets from being obtained by others. Finally, it is seriously limited in that anyone able to obtain the secret may then complete a separate authentication process to the same identity, and gain the same level of access as the user.

7.1.2 Strong Authentication of Identity

In order to alleviate these limitations of weak authentication, the concept of strong authentication is introduced. With strong authentication, there is no requirement for the transfer (and thus disclosure) of the authentication secret to the verifier; instead, the user wishing to authenticate their identity does so by proving their knowledge of the authentication secret, rather than revealing it. By no longer divulging the secret on each authentication attempt, strong authentication techniques can also offer protection against various attacks involving the interception of the secret, or the replay of past authentication sessions [225]. An example of such strong authentication is the Secure Remote Password (SRP) protocol, which is designed to provide verification that a user has knowledge of a particular password, without the password being revealed, or held on the verifying system in a form which could be used to determine the user's password [122].

7.1.3 Trust in Authentication

One remaining limitation of strong authentication is the requirement for a trusted verifier. While strong authentication protocols like SRP remove the need for users to disclose their raw passwords in the process of authenticating their identity, the actual software carrying out the verification must be trusted not to produce either false positive, or false negative, statements of validation [122]. In a false positive scenario, the verification server asserts that a user passed verification for an identity when they did not actually have knowledge of the password or secret, and in a false negative scenario the verification server wrongly asserts that a user failed authentication to their identity. Both of these scenarios are problematic in the authentication of identities — in the former, an unauthorised user may be wrongly authenticated to an identity other than their own, and in the latter, an authorised user may be denied the ability to authenticate to their own legitimate identity. Similarly, the service provider also needs to securely store user credentials, or the representations of those, such as hashes, used to verify credentials.

7.1.4 Federation of Authentication

Recent advances in authentication technology have often focused on federated, delegated authentication, where a user may authenticate to an identity within a given service by identifying and authenticating themselves to an external, unrelated third party service. Examples of this kind of identification and authentication federation include OAuth2 and OpenID. In these scenarios, a user wishing to identify and authenticate themselves is asked to select the external service they will authenticate with. The user is then redirected to that service to carry out its login process. If it succeeds, the external service will return a cryptographically signed statement affirming that the user has authenticated, and thus verified their identity correctly. This signed response is verified by the first service, and if valid, is regarded as a valid assertion that the user has correctly authenticated.

7.1.5 Limitations of Delegated Authentication

Schemes such as OAuth2 allow services to separate themselves from the handling of user authentication, and for the establishment of third party identity and authentication providers. A user therefore no longer needs to trust the original service provider to authenticate their knowledge of a password and other secret properly, and to store that password securely or in a manner which cannot be compromised by others. One fundamental limitation of such schemes, however, is that while offloading the process of authentication to a third party identity and authentication provider, the need for trust is also similarly offloaded. Under a federated, delegated authentication system, users must fully trust their identity and authentication provider to only ever issue valid assertions, and to never be tricked into issuing a false assertion.

In terms of dependency upon third parties, use of federated and delegated authentication therefore introduces an element of intentional centralisation around the login process. Since third party services are not required to store verification information for user authentication, one advantage of this is that security breaches do not threaten to compromise user credentials, as there are no passwords or equivalent verification materials stored or available. A disadvantage of this approach, however, is that the service's user identity and authentication becomes entirely under the influence of the third party providers, and it is no longer possible for the service in question to ensure authentication requests are legitimate, other than by trusting the third party authentication provider.

In the event of a malicious entity compromising or coercing the third party authentication provider, or one of its privileged employees with access to the source code or backend database, it would be possible for the authentication provider to make a legitimately signed false statement of authentication to a given identity. This would allow an unauthorised user to gain access to an identity they cannot legitimately authenticate to.

7.2 Enforcing Authentication

One of the challenges of authentication, as described so far in Section 7.1, is that it is fundamentally built upon a premise of an attestation. If a typical web-based application which requires authentication to a user account is considered, the result of authentication is a boolean confirmation of whether or not a given session is permitted to make use of the service under a given identity. Fundamentally though, as this attestation is simply the boolean output of whether or not the action is permitted, the authentication process can be bypassed by the service operator, or someone with access to their servers, to allow a user to access a given account. This is based on the same premise as the weakness on delegated authentication, as described in Section 7.1.5.

This bypass of authentication could be a result either of malicious actions, or of unintentional yet non-malicious actions, as seen in the case of Dropbox [8], where, as a result of an unintentional code change, for a period of around 4 hours it was possible for any user to log into any user account without knowing its password, and gain full access to the files stored within that account. Scenarios such as this highlight the importance of the link between identity and authentication — without adequate and proper authentication taking place, the identity, from the perspective of the service, is unable to be trusted or relied upon. To create a secure system, authentication and identity must therefore be sufficiently linked, such that a failure of either will prevent use of the service. Without this, it is possible (as shown in the case of Dropbox) for an error or malicious action to result in the unauthorised use of an identity, putting the confidentiality of user data at risk.

7.2.1 Mandatory Authentication

Taking into consideration the risk of an error in the authentication process resulting in the improper handling of user identities within a web application, the concept of mandatory authentication can be considered, whereby the process of authentication is embedded into the use of an identity, such that it would be impossible for a service provider to incorrectly authenticate a user and grant access to data. This would allow for a service whereby users need not trust the service provider, or indeed a third party authentication provider, to carry out authentication correctly, since the use of an identity would require integral authentication through knowledge of a secret, or other authentication factor.

An example of this kind of authentication is the "self authentication", as proposed by MaidSafe, and detailed in Section 2.5.3. When using self authentication, there is no separate stage of authentication required prior to the issuance of a request; rather, the request itself is cryptographically signed by the holder of the identity, such that the request may be verified by anyone knowing the identity, which is the public key. This allows for the creation of an effectively stateless service, where no state information (identity, authentication state) need be held by the service. By removing this requirement for session state to be held, it is therefore no longer necessary to trust the service in question to carry out user identification and authentication. Instead, by verifying that data being sent to a service by a user is signed by the corresponding identity key, it is no longer necessary for a third party (such as the service provider) to be trusted to carry out authentication; any third party, trusted or otherwise, may verify the signature on the data to be sure it originated from the user it claims to belong to.

7.2.2 Identity-based Cryptography

The process of self-authentication may be considered a form of identity-based cryptography [226], whereby some kind of public information about a user (effectively part of their identity) is used as the user's public key. In conventional identity-based cryptography, however, there is a requirement upon a highly trusted third party, which generates private keys for users based upon their generator secret. In this manner, a user may verify a message from another user, simply by knowing their identity (such as their employee number or email address). The obvious limitation of a scheme such as this is the trust placed in the trusted key generator, which is capable of generating the private key for any given public key.

The process of self authentication presents an alternative to this approach, by considering a user's identity to be their public key (or its cryptographic hash). Unlike identity-based cryptography though, it remains necessary to verify the key presented by a user on the first occasion it is used, to ensure that the correct mapping between user and public key is established, since the public key acts as a form of consistent pseudonym. There is no requirement for trust in a third party, since each user is able to create their own identity by simply generating a new private key, and publicising the public key component of it.

The conflation of identity and public key provides a verifiable form of security — the process of exchanging identities also serves as an exchange of the user's public key. This facilitates secure product designs by default, since by default a user's identity facilitates encrypted, authenticated communications using asymmetric cryptography.

7.3 Biometric Authentication

Biometric authentication has been one proposed solution to the challenge of authentication and identification of users. One particular implementation which has gained significant traction and interest in recent years has been fingerprint authentication, through the use of an on-device fingerprint reader. This section shall explore practical security considerations as to the use of biometric authentication, and show that fingerprint authentication does not present any viable solutions to the challenge of authentication.

7.3.1 History of Fingerprint Reader Usage

Fingerprint readers are, without doubt, one of the must-have features on almost every smartphone launched in 2015 or 2016. The Samsung Galaxy range of flagship devices have featured a fingerprint reader since the Note 4 and S5 [227]. Sony's latest Z5 range include fingerprint readers across the range [228]. Indeed, even newer entrants to the market such as OnePlus are including a fingerprint reader on their handsets [229].

Smartphone fingerprint readers are used typically to implement biometric authentication. Biometric authentication is a means of authenticating a user based on making measurements of one or more physical characteristics — in this case their fingerprint.

Fingerprint authentication on personal computing devices is not an entirely recent concept, having previously been seen on Thinkpads and other enterprise laptops, and even on some high-end Personal Digital Assistants (PDAs). A more detailed history of the use of biometrics and fingerprints has been presented by Corcoran [230].

The first widespread, consumer-focused smartphone implementation of a fingerprint reader was seen on the Motorola Atrix, launched in early 2011, which supported using the inbuilt fingerprint reader as a means of the user authenticating to their phone's lockscreen [231]. Following the introduction of the iPhone 5S in September 2013, featuring a fingerprint reader integrated on the home button, fingerprint readers within smartphones became much more popular; The HTC One MAX was released in October 2013 with a fingerprint reader, and subsequent products from many manufacturers followed suit.

With such widespread adoption, it is important to investigate the user factors which may have resulted in this increase in popularity, as this may offer an insight into what users feel important on their smartphones, and what they are willing to accept.

7.3.2 User Attitudes and a Need for Convenience

The premise behind fingerprint authentication on mobile devices is typically as a replacement for the password or PIN. Good passwords require users to follow a multitude of rules, ensuring the length, complexity and uniqueness of every password they use. Remembering unique passwords for an ever-increasing number of services places a significant demand upon users, and leads to more easily guessable, or re-used passwords. Users now carry out 50% of their password entry operations on smartphones, where special characters are difficult to type, and long passwords are inconvenient [120].

A key attraction of biometric authentication is that it allows users to move away from passwords, both for use in authenticating to third parties, and for unlocking their own physical device. This eliminates the requirement to enter passwords, and avoids the inconvenience of forgetting passwords.

There is clear indication from previous studies that users are aware of, and willing to use, biometrics — in their 2005 survey, Clarke and Furnell found that 83% of surveyed mobile phone users would be willing to use biometric authentication [232]. Indeed, of those aware of the existence of fingerprint authentication, 99% were happy to use it.

This is in clear contrast with iris recognition, where only 70% of those who were aware of it were happy to use it.

In contrast, an earlier survey from 2000 [233], which focused more generally on authentication, rather than specifically on mobile phone authentication, found 67% of surveyed users were willing to use fingerprint authentication. While this would indicate that either user attitudes towards fingerprint authentication have changed with time, or that users consider mobile authentication a special case, a 2007 study on the uses of authentication technologies [234] found that only around 40% of users surveyed agreed biometrics were useful when accessing a computer, in contrast with 66.1% when considering financial transactions.

It is worth noting that these surveys were carried out prior to the recent widespread adoption of smartphones. Nonetheless, it is clear that users are willing to use fingerprint authentication, and that the increased portability of smartphones, combined with the large quantities of personal data stored within, is potentially a driver for the uptake of the technology. Harbach *et al.* showed that the perceived inconvenience of a secure lockscreen on a smartphone was a factor in around one third of people not enabling one, and with an average of 48 unlocks per day, there is a clear argument for convenience of unlocking frequently used devices like smartphones [235].

7.3.3 Biometrics in Authenticaton and Identification

Fingerprints, and biometrics in general, present users with a simple alternative to PINs or passwords, to which they are accustomed. Fingerprint authentication is often viewed by users more favourably than alternatives, due in part to the user perception that biometrics are the most secure form of authentication [234]. In particular, there is a common belief that fingerprints are secure in part due to their relative uniqueness, and their use in criminal justice, which may also add to their positive perception as highlighted above. This does raise an important distinction though when considering the use of fingerprints — the needs of an identification system are somewhat different to those from an authentication system.

In a biometric identification system, the goal is to reliably ascertain who an individual is, based upon a comparison of measurements taken from a sample, which are then matched against previous measurements. For this to work correctly, each individual in a population should be uniquely defined, and should be recognisable in future against a previous measurement. Therefore, there is considerable focus on the uniqueness of the characteristics. For example, in a criminal investigation, the aim of forensic fingerprint analysis is to recover the fingerprints of individuals who may have been present at a crime scene. Biometric identification is then carried out to ascertain if this recovered fingerprint matches that recovered from any other crime scenes, or from individuals known to have previously committed crimes.

In contrast, a biometric authentication system is designed to allow an individual asserting their identity to prove this assertion, based upon their ability to provide biometric measurements in keeping with previously enrolled values. In the authentication scenario, a rapid result and a low false-positive rate are desirable. One consideration is that a strong authentication process, per the definition from the European Central Bank, is required to be non-reusable and non-replicable, to prevent re-use of a previously valid authentication session which may have been observed. There is however an exception made for authentication based on biometric factors, since it is inherently based on static measurements of a person. The distinction between identification and authentication is also discussed in [230].

7.3.4 Fundamental Limitations of Biometrics

7.3.4.1 Static and Unchangeable

The most fundamental limitation of fingerprint-based authentication is that our fingerprints are the ones we were born with, and the ones we keep for life. They are, by virtue of being part of us, unchangeable. This is an advantage in one sense, as a user cannot 'forget' their fingerprints in the same way an infrequently used password can be forgotten over time. For many users, the convenience of not forgetting passwords is a significant draw of fingerprint authentication.

7.3.4.2 Irrevocable Identifiers

By virtue of being static in nature, there is no effective means of revocation. If your fingerprints are compromised by some means, there is no way you can prevent the compromised copies from being re-used in future. This is a limitation of the process of fingerprint authentication, since ultimately the verifier is expecting to see the same fingerprint on each occasion. Breaches of fingerprint data are now no longer a hypothetical situation, given the recent theft of 5.6 million US federal government employees' fingerprints from the Office of Personnel Management (OPM) [7].

7.3.4.3 Easily Observed and Captured

Fingerprints are also easily captured without the subject being aware, both with and without physical contact. As highlighted in 2013 following the highly publicised "breaking" of Apple's new Touch ID feature, a latent fingerprint was captured using a high-resolution photograph of the glass touchscreen of an iPhone. It was then used to create a mould that could form an artificial fingerprint, capable of unlocking the phone [236]. While a relatively in-depth process, this highlights one fundamental risk of relying on fingerprint authentication on smartphones — they are held and touched by the user in daily operation, and their large glass surfaces act as a magnet for the user's fingerprints.

Additionally, it was recently shown that fingerprints can also be captured without physical contact. A series of high resolution photos, including one from a press release, were used to recreate the fingerprints of the German defence minister [237]. Indeed, this is not the first occasion in which a German politician's fingerprints have been publicised; in 2008, an index fingerprint was obtained and reproduced from a water glass used by the German interior minister during an event at a University, resulting in over 4000 copies being made onto plastic foil capable of being used on various fingerprint readers [238].

7.3.4.4 Their Pervasive Nature

Another property of fingerprints is that they are static, and cannot easily be 'turned off'. As you go about your life, you are leaving a trail of fingerprints around. With the rise of fingerprint authentication, this could be considered tantamount to leaving a trail of sticky notes containing your username and password to every account you have, every time you touch something.

If a password is compromised or known by someone else, it is relatively straightforward to change it, therefore revoking it, with the biggest inconvenience merely having to memorise a new password. Since they remain constant at all times, this isn't possible with fingerprints. Our inability to effectively control where our fingerprints are left behind is a significant concern. You can be careful to only type your bank password in the privacy of your own home, on a system with no keyloggers, with the curtains closed to prevent onlookers, but if you use a fingerprint to authenticate with your bank (either directly or indirectly), you cannot avoid leaving those fingerprints around. There is no concept of security level with fingerprints in the same way that one can use a single low-security "throw-away" password for uninteresting accounts which don't contain any personal data of value.

7.3.5 Limitations of Fingerprint Sensing

Smartphone fingerprint sensors have been subject to a variety of high-profile attacks, where fake fingerprints made from a variety of materials designed to have properties similar to human skin have been accepted as valid fingerprints. Indeed, these techniques are not outwith the practical reach of private individuals [236].

More fundamentally, a fingerprint sensor within a computer system is typically designed to convey a measurement of an individual's raw fingerprint to another component of said computer system, responsible for either deriving a cryptographic key, or unlocking an existing cryptographic key held securely on the device [230]. Therefore, by identifying the input format expected by the key storage or computation module, it is possible to present a falsified (or previously-captured) fingerprint reading "on the wire", thus bypassing the need to handle the creation of a fake physical fingerprint.

7.3.6 Legislatory Concerns

7.3.6.1 Fingerprinting in Criminal Process

Within most countries in the world, there is a presumption of innocence for those accused of crimes, until they are convicted at a trial in a court of law. Until that point, they remain innocent and not having been convicted of any wrongdoing. In many jurisdictions, those who are under arrest, and have not been charged with, or convicted of, an offence, may be required to give fingerprints, which can be held on a database, for the purpose of identifying any linked crimes an individual may be responsible for [239].

This process, by its definition, involves the capturing of a record of an individual's fingerprints. Since fingerprints are static and irrevocable, this individual's fingerprints are now potentially on file indefinitely. If fingerprints are used as a secure means of authentication, this is equivalent to being required to hand over a full list of all your past, present, and future passwords, simply as a part of the investigatory process.

While there may be legal procedures through which individuals can appeal to have their records removed if they were not convicted of an offence, it will never be possible for that individual to be sure that their fingerprints no longer reside on a database somewhere. The same applies to those travelling to a country which stores fingerprint records of those entering as a matter of routine, such as the USA under the OBIM program (formerly US-VISIT) [240]. Large databases are not impenetrable to unauthorised access either, as was shown in the OPM breach mentioned above.

Fingerprints may also be handled differently from a legal perspective than something which is known to a person, such as a password or PIN. In the US state of Virginia, a judge found that requiring the disclosure of a password or PIN would be in breach of the 5th Amendment, but that requiring a person suspected of committing a crime to use their fingerprint to unlock a device was constitutional [241]. It highlights an interesting situation on some devices, such as the iPhone, where a fingerprint can only be used within 48 hours of the last successful fingerprint login, after which the PIN must be used.

7.3.7 Current Implementation

7.3.7.1 Weaknesses in Smartphone Implementations

Today's consumer devices featuring fingerprint authentication technology typically make use of the ARM TrustZone Trusted Execution Environment (TEE), which allows for isolated *secure world* code to be executed on the regular CPU, separate from untrusted user code, such as that of a mobile device's operating system.

Recent research by Zhang *et al.* has nonetheless highlighted the problem of poor implementational security of fingerprint readers on many mobile devices. In their paper, a number of security issues with implementations of fingerprint sensors in mainstream phones were identified [242]. In the most extreme case of the HTC One MAX, the user's enrolled fingerprint was stored in a world-readable file. This meant that any unprivileged application running on the phone could read the file containing a user's fingerprint, without the user being aware.

While the established best practice for implementation of fingerprint readers involves the use of the ARM TrustZone to hold, validate and handle all fingerprint data, Zhang *et al.* highlighted that even with this in place, there have been exploits against TrustZone, and the fingerprint reader device is often exposed to the regular, non-TrustZone operating system of the mobile phone. This allows the fingerprint reader to be accessed by software running on the phone, provided it is able to elevate its privileges sufficiently to do so.

In addition, there have been numerous publicly documented exploits of TrustZone technology [243, 244, 245]. All of these allowed for arbitrary code execution within the secure environment, and the latter specifically gives a proof of concept to show how the user's raw fingerprint can be captured and retrieved from the reader, despite only code running in the TrustZone being able to read from the fingerprint reader on the device in question.

7.3.7.2 Implicit Trust in the Reader

Another, more general consideration with today's implementation of fingerprint authentication is that of trust of the capture device. Since, by definition, fingerprint data is constant, it is necessary for the reader or capture device to be trustworthy, and not store or transmit it for use or storage by unauthorised parties. This raises the questions of who is authorised to receive the data, how the biometric data may be used, and the manner in which it may be stored and processed. Specifically when a device holds biometric data (such as a smartphone), a question arises over if the company who manufactures the smartphone has, or should have, any right (or ability) to access that data. In the case of Android devices, for example, there is also the question of whether or not Google (the developer of the core operating system) has the ability or right to access the data.

A rise in the use of fingerprint authentication on smartphones would also likely fuel a rise in the use of fingerprint authentication in other areas. For example, Poland has installed bank ATMs featuring fingerprint authentication since 2010, where a fingerprint is used in conjunction with a PIN to withdraw cash [246]. Significantly, this requires users to provide their fingerprints to an unverifiable device operated by a third party. Fake (or real, with unauthorised modifications) ATMs have been a popular way for criminals to “skim” cards and obtain PINs via fake keypads and card readers.

If users become comfortable with providing their fingerprint to equipment requesting it, without being familiar with it (to identify signs of tampering or illegitimacy), they may find their fingerprint data is stolen by criminals. While the same is completely true of bank card numbers and PINs as used presently at ATMs, there is little long-term impact of such details being compromised; the bank freezes the account and reverses the transactions, and issues a new card to the account holder, who sets a new PIN. In the case of biometric authentication, it is not possible to change or revoke the biometric.

7.3.7.3 Other Potential Uses

Biometric data is potentially of huge value to advertisers and other businesses, as it allows for theoretically globally unique identification of users, simply based upon their use of a product or service. If fingerprint readers become a common feature of consumer electronic devices such as smartphones, undoubtedly the question over rights to use such data will emerge and need answered.

Whether it is legal, ethical or acceptable for fingerprint data to be used to pervasively track a user is a question which should be answered before such technology becomes widespread, otherwise we may find ourselves in a situation like we face with internet-based services, where users have relatively limited technical controls and restrictions over the use and sharing of their personal data, and websites carry out widespread tracking of user activity and actions across the wider internet.

The ability for an advertiser to tell with certainty that the current visitor to a website, or user of an application, is the same one as in a previous browsing session, would be of incredible value — this would persist across devices and browsing sessions. It would also be effective against attempts to prevent such tracking, such as a user clearing their cookies. While the suggestion not to provide fingerprints to such websites may well be the obvious one, ensuring this is enforced with technical (rather than legislatory) measures is essential. With current fingerprint reader implementations generally “black box” systems, not open to scrutiny by researchers or experts, this is difficult to achieve.

7.3.7.4 Lack of Ability to Verify Actions Being Carried Out

In contrast to a PIN or password, where a user is prompted to enter a particular one for a given service or action, fingerprints remain constant between services. This means that the contextual information as to the action being carried out following fingerprint verification is critical. Since the same fingerprint may be used to unlock a device, as well as authorise a high-value bank transfer, it is important to ensure users have a reliable and trustworthy way to understand the operation they are approving via their fingerprint.

Entering a PIN requires a user to understand the action they are authorising — presuming a user follows good practice and doesn't have the same PIN on all of their bank cards, they can easily detect that they are carrying out a transaction on the wrong card due to the PIN being rejected. Likewise, while not fool-proof against malicious attack [247], the screen on an EMV chip-and-PIN payment terminal confirms the value of a transaction being carried out, or the recipient and value of a transfer. On a smartphone featuring fingerprint authentication, simply providing a fingerprint is sufficient to carry out a variety of operations. These can extend from merely unlocking the device, to logging into an app or website, to initiating a bank transfer. Indeed, smartphone apps from major banks now allow for the use of fingerprints to authenticate transactions [248].

7.3.8 Future Considerations

7.3.8.1 Avoiding the Reader Completely

With the rise in smartphone fingerprint sensors, it is interesting to note that, while early devices with such sensors (such as the Motorola Atrix) featured their sensor on the rear of the device, where it could be avoided or covered by a case, more recent implementations (such as the iPhone and Samsung's Galaxy range of devices) feature the fingerprint reader on the front, within the device's physical home button. This presents usability benefits for consumers, since authentication can be carried out using a button they already use for other tasks. Additionally, for the purpose of unlocking the device, the same button which was used to wake the phone can then be immediately used as a fingerprint reader to verify the user's fingerprint. On the other hand, this also makes it easier for a user to unintentionally authenticate a request, simply through over-familiarity with the process.

Continuing this trend, it is conceivable that in the future, the need for a fingerprint reader in itself may be eliminated, given a recent patent application by Apple [249], to include a fingerprint reader within the screen. At that point, and arguably also today, with the reader a component of one of the major buttons on the phone, the question arises over if a user has a choice as to whether they wish to be fingerprinted or not while using a device. While users can avoid using the reader for the purpose of authentication, it is much more difficult with the current “black-box” style fingerprint authentication systems to verify that the fingerprint data isn’t being read or stored. With sensors embedded in screens, it may not be clear to a user when they are authenticating a request, since the authentication process may no longer be a clear distinct action, thus bypassing the careful consideration that should be taken before proceeding.

7.3.8.2 Fingerprint Payments

The latest, and potentially one of the most visible, consumer application of fingerprint-based authentication on smartphones is for the authentication of payments carried out via a mobile device. By placing their smartphone against a contactless reader, a user is able to select the card to use for payment, and authenticate the payment by simply placing their finger against their smartphone’s fingerprint reader.

Early implementations which are seen on today’s consumer devices, such as Apple Pay, appear to have a number of weaknesses, as exhibited in their own demonstration. Specifically, there is no authentication of the transaction amount visible — as seen in their product demonstration, an Apple Pay user simply knows that they are being asked to approve a transaction with the selected card, but there is no indication of the value of the transaction being carried out [250].

Taking into consideration the design of smartphone-based payment systems, which are now deployed and operational in the USA and UK, there is a risk of early users being victims of fraud, as a result of innovative fraudsters, given the reliance of these systems on fingerprint authentication. Even putting aside the limitations of fingerprints being unchanged and potentially known to third parties, and the ease with which they can be captured, a fingerprint reader is ultimately used to authenticate to a “trusted” area of the smartphone, often based on ARM TrustZone technology, as discussed earlier.

If the contents of this TrustZone were to be compromised, the user’s fingerprint would most likely no longer be necessary in order to authorise transactions on behalf of a user. The presentation of a permitted fingerprint is used by the semi-isolated Trusted Execution Environment (TEE) to permit the use of cryptographic keys, which are themselves only accessible by the TEE. In the event of the TEE being compromised (as discussed earlier), these keys may be exfiltrated from the device, or otherwise abused by a malicious user (such as by forcibly enrolling a new fingerprint).

7.3.9 Potential Mitigations

Despite many of the potential risks and challenges of the use of fingerprints in secure authentication, it is clear that consumers feel it is secure enough, and products incorporating fingerprint readers are now reaching the market in significant quantities. In this section, some ways in which these risks can potentially be mitigated or reduced are considered, to allow for a practical solution to the clear user demand for a simpler means of authentication.

7.3.9.1 Legislation

Firstly, strong legislation is necessary to govern how biometric information may be used, and shared. Where such legislation exists, it often covers only government or official use of biometric information [251], rather than commercial or third party use of, and gathering of, biometric information.

When a user voluntarily provides their fingerprints to a piece of consumer electronic equipment, they are no longer engaging with a legislated entity. Indeed, in many jurisdictions, the handling of electronic or personal data (which may include biometric data) is left to self-regulation and loose oversight, rather than legislation [105].

Given the unique way in which biometric information cannot be changed, legislation governing the technological protection of biometric data is necessary, to ensure that consumer technology utilising it is designed to reduce the risk of compromise as much as possible.

7.3.9.2 Transparent Implementations

In order to mitigate many of the risks of current implementations (such as TrustZone exploits and similar), it would be advisable for implementations of biometric authentication to be designed and documented publicly, with all relevant source code as to the operation of the authentication mechanism made available for review. With fingerprint authentication set to become near-ubiquitous in the short-term, there would be considerable benefit in ensuring that the technology is secure, on account of the hesitations people hold about the use of biometric authentication, and the significance with which end users place on trust and resistance to attack [252]. Ensuring that implementations are transparent and open to independent scrutiny would facilitate verification of correct implementation, and the identification of security weaknesses. While it could be argued that such disclosure would make attacks easier, a lack of source code has not held researchers back in finding vulnerabilities in TrustZone and other fingerprint reader implementations, as discussed earlier.

7.3.9.3 Trusted Software

If a product offers a consistent and predictable user interface for the request of fingerprint authentication, it is important to ensure that this interface is trustworthy. For example, it is critical that the application or service requesting identifying information is clearly and correctly identified to the user, to prevent social engineering attacks, or falsely generated prompts from overriding the system prompt to change the appearance of the prompt (making it appear a different application is requesting authentication).

At each opportunity, the user should also be clearly presented (using a trusted software implementation, which is again open to scrutiny and security testing by independent researchers) with a summary of the action being carried out at each point. A separate cryptographic key should be used for each application using biometric authentication, to prevent a rogue application from generating a valid authentication message in response to its own request, which would be accepted by another service as an attestation that the user had agreed to an operation. This would be a risk in a scenario where a service accepted a signed random value as an attestation — another application could request the same random value, and replay the token, unless a unique key is used for every application.

7.3.9.4 Avoidance of Over-use

One factor identified here is that the over-use of fingerprint authentication may well pose a risk. Consumers seek convenience, and the convenience of fingerprint authentication is attractive, compared with the task of typing lengthy passwords on a small on-screen keyboard. Despite this, repetitive authentications result in people becoming lazy, as is seen in their use of short or simple passwords for which they are asked for regularly. If fingerprint authentication is over-used, it seems likely that people may become overly comfortable with simply approving everything that is requested, rather than validating the precise request. Especially when a fingerprint reader is located on the home key of a product, the natural reaction will be to approve the action, rather than to scrutinise it further and verify that it is indeed the action which should be carried out. By encouraging users to pause and consider the request, perhaps even enforcing this via a short on-screen time-out, this would go some way to ensuring that users are aware of the action they are providing authentication for.

7.3.9.5 Awareness of Risk

It is also important for service and application implementers to be aware of the risks of fingerprint authentication, particularly around those whose fingerprints may be known to, or have been captured by, third parties. While today's payment solutions allow fingerprint authentication as a means of proof the card-holder is present, this ultimately relies on the integrity of the TrustZone implementation used to hold these keys. If these keys were extracted, or a man-made fingerprint was presented to the reader, the proof that a customer was present to authorise a transaction is less robust. With the ease with which an unwilling party can be compelled to give their fingerprints, it is also likely that people may be forced to unlock fingerprint-authenticated equipment against their will, to authorise transactions or simply for their fingerprints to be captured for future use.

7.3.9.6 Plan for Compromise

Finally, in light of the previous point, it appears necessary to begin to plan for a future where fingerprint and other biometric authentication is readily subverted by malicious use of, or threat of, force. While the same is true for today's passwords and PINs, users are always capable of selecting and using a new password. Early adopters of biometric authentication technologies will be at risk of emerging threats, and we should be prepared for a time where people's fingerprints are widely known. For this reason, it is important to consider this risk in future, when deploying biometric technology, and for companies relying on it to be aware that the presence of a seemingly biometrically-verified signature does not necessarily indicate the user has agreed or given their consent. This may also have implications on the legal status of biometrically-authenticated signatures.

7.3.10 Conclusions of Fingerprint Authentication

It is clear from the above that fingerprint authentication, and biometric authentication in a more general sense, is based upon a premise whereby a user's biometric information cannot be spoofed or otherwise retrieved by an unauthorised individual. State-of-the-art implementations are built around a model whereby the fingerprint reader requires a trusted path to a secured area of the main CPU, where the verification of biometrics may take place, and operations carried out if authentication succeeded. Implementations have, however, been shown to not carry this out correctly, indicating that the assumption a TrustZone-based implementation offers adequate security is unsafe. Coupled with the inherent nature of fingerprints (and other useful biometrics) as unchangeable, it is clear that while useful for identification, they do not offer the necessary properties for use in secure authentication, since there are already large numbers of people whose fingerprints or other biometrics have been captured and are being stored, thus negating any security or privacy for these individuals, were these records to be used in future. For this reason, biometric authentication should not be used to secure access to long-term secrets or keys, due to the demonstrated regularity with which vulnerabilities are found in implementations, which would prove catastrophic for users relying on this to store their information securely and privately. A better solution would allow for the revocation of keys, the ability to have more than one set of keys, perhaps for different types of data or actions, and would prevent a passive user from unknowingly giving another party access to their authentication credentials.

7.4 User Credentials and Key Storage

As described previously, usernames and password-based weak authentication remains the most popular way of carrying out authentication with a service. Recent high-profile security breaches have highlighted a number of human factors concerning the use of memorised authentication credentials. While a variety of best-practices exist around password security, previous studies have highlighted the extent to which users eschew guidelines and recommendations, simply on account of the impracticality of implementing them. One specific example is the advice to use different password on every service, so if one website is compromised, no other accounts are compromised — despite this, many users continue to select the same password for every service, on account of the difficulty in memorising and using such a number of passwords. Most password-based logins also retain the disadvantages of generic weak authentication schemes; since the password is disclosed during login, it can therefore potentially be socially engineered or otherwise observed, and then re-used.

Existing technologies such as SSL and TLS client authentication certificates can be used to implement strong authentication, whereby the process of establishing a TLS connection with a remote server verifies the authenticity of the client's certificate, based upon its signature. It is therefore possible for a certificate authority (CA), perhaps run by the service, to sign user certificates based upon a user's verified identity, and then require this certificate to be presented in the future during authentication.

One limitation of this process, however, is that it places a requirement upon a centralised certificate authority, which could be compromised or compelled to issue a valid certificate for a given identity to another user. There are also a number of security considerations around the use of client certificates in the browser, such as the ease with which malicious software (or social engineering) can be used to obtain a user's private key files from their filesystem, and exfiltrate them to a malicious party. One potential mitigation against attacks like this is the use of a separate, more trusted, key storage device, which interacts with the authentication request and securely holds the user's keys, without the ability for them to be disclosed or extracted by third party software. This forms the premise of smartcard-based authentication.

7.5 Smartcard Authentication

The term smartcard is typically used to refer to a portable and standalone module, capable of the storage of, or proccessing of, data separate from a host device which presents it with a request. Smartcards are seen, amongst other implementations, on modern banking cards, as a means of securely authenticating transactions, in GSM mobile phones as a secure storage area for the cryptographic keys used to establish and assert a given subscriber's identity to the mobile network, and in loyalty cards and pre-payment cards for telephone use or utilities.

Smartcards are typically designed to a high specification to resist attacks such as power analysis and power glitching. The intention is to ensure that when under active attack, data held within the smartcard cannot be determined by any external attacker, and that the operation of the smartcard cannot be influenced by the external attacker, except in manners that were designed (such as legitimate user input). Smartcards are therefore ideal for the secure usage of cryptographic keys, since they are designed to be resistant to attacks which may be used to retrieve those keys directly from the memory on the card. The smartcard can carry out the cryptographic processing requiring the key, therefore ensuring that strong authentication is enforced — since the smartcard will not reveal or allow others to access the raw key, a successful authentication on a well-designed, replay-resistant authentication service therefore proves that the correct card was present during the authentication attempt. This forms a second factor of authentication (something you have), which can be combined with a PIN or other secret, that is used as a first factor (something you know). By having the smartcard validate the PIN, it need not be revealed to any third party, and the smartcard can be protected against use by unauthorised persons. Therefore, carrying out a single authentication request can indicate that two factors of authentication were validated correctly, provided the issued smartcards can be trusted to carry out this validation correctly, and the smartcard response was correctly validated.

7.5.1 Existing Smartcard Authentication Schemes

A number of existing standards and implementations allow smartcards to be used for authentication. PKCS#11 offers a standardised interface through which software may carry out cryptographic operations (such as signing and encrypting), without revealing the keys held within the smartcard [253]. The Personal Identity Verification (PIV) interface, from *NIST SP 800-73*, likewise defines a standard for identity verification and authentication using keys held on a smartcard, for use in the US military and federal government.

Previous work has considered the security of the PKCS#11 techniques for the binding or wrapping of cryptographic keys, and highlighted the complexity of implementations. Hertl reviewed a number of attacks within PKCS#11 [254], and the design of the proposed solution mitigates against each of these. The impact of these attacks is discussed in more detail in Section 7.8.9.

These authentication protocols are fundamentally built upon the principle of using a smartcard for secure cryptographic operations, without disclosing the keys involved. One limitation of these implementations, however, is that they are typically designed to allow for the use of only a single identity on a given smartcard. This is also seen in other implementations of smartcards for key management or authentication, such as on the OpenPGP card, where only one set of keys can be held on the smartcard.

7.5.1.1 Multiple Identities

While holding a single identity on a smartcard is advantageous in situations where users should only have access to a single identity (such as in a workplace handling sensitive information, or gaining access to secure facilities), in more generic use-cases there are advantages to user privacy of having multiple independent identities, from the perspective of third party services being used. For example, a third party service provider may record the identities of users accessing information on a service. A user may wish to access information without disclosing their identity to a service provider, perhaps to read information about a medical condition, or perhaps simply because they wish to access it without a permanent record of accessing the content being tied to their identity. In these scenarios, the ability for a user to create multiple identities may be advantageous for user privacy, as it allows users to switch between identities as they desire, retaining the convenience of being logged in and authenticated to a service, without having all their activity recorded against their original account. Within the context of personal and business use, implementations such as Android Work introduce separate profiles on mobile devices, to allow corporate data to be kept entirely separate from personal data, even though it may be held and accessed on the same device.

This level of isolation is therefore of utility to users, and is desirable to privacy-concious users, who wish to control what information may be associated with their identity by third parties. Due to the limitations in storage capacity of smartcards, however, there is clearly a finite limit as to the number of separate identities which may be held on a given smartcard. Larger capacity smartcards are also more expensive than those with lower storage capacities, on account of the increased protected memory being required on the card. In comparison with other microcontroller-based systems, it is notable that storage capacities of smartcards typically are significantly lower than conventional unsecured microcontrollers, on account of the cost and complexity of securing such memory. By way of example, a 40KB memory dual-interface Javacard smartcard is available for sale at £5.65 per unit.

7.5.1.2 Identity Size

In order to establish a secure identity, contained on a smartcard, a minimum asymmetric key size of 2048 bits was selected, on account of the Certificate Authority and Browser Forum (CAB) requirement that all RSA keys issued after 2011 have a minimum modulus of 2048 bits. To offer defence against key-misuse attacks, where a user is asked to sign a given message, where signing is mathematically the same operation as decrypting, such as in RSA, the precaution of creating two identity keys is beneficial. One key may be used exclusively for signatures and attestations, and the other key may be used solely for decryption of incoming messages. This indicates that a user identity contains two 2048 bit keys, in a representation which allows them to be unambiguously distinguished by software using the identity.

This form of identity identity satisfies the properties of an identity, as described earlier in this chapter, since the user's public key remains consistent throughout time, and can be distinguished from all other users' public keys, in the absence of two users generating the same private key. If this were to happen, however, this would indicate insufficient entropy in the random number generation stage of key generation, and the keys would likely be broken with relative ease.

Since smartcards typically contain only a few kilobytes of persistent memory (and often even less transient random-access memory), it is clear that the storage requirement for an identity, while small, is significant when considering a smartcard of limited storage capacity. For example, some programmable smartcards may ship with as little as 8 kilobytes of persistent memory, which must be shared between the program code, and the keys held on the card. Using 512 bytes of this for a single identity will quickly make identities difficult to scale, and require the re-use of identities in order to hold them all on a single card.

To grant users privacy, however, as discussed earlier in this chapter, it is desirable for users to be able to present multiple identities, therefore granting them the benefits of a consistent identity with which to establish a reputation, while allowing the separation of different activities under separate identities. To achieve this, the ability to hold keys without being constrained by the capacity of the smartcard is desirable.

7.6 Secure Smartcard Identity Storage

In order to remove the constraint of smartcard capacity, thus making it possible for a single user to access an effectively unlimited number of identities in a practical manner, a means of securely offloading keys from a smartcard us presented. This implementation was designed around a robust threat model, designed to address the secure usage of sensitive data on the smartcard. By way of comparison with the current state-of-the-art, the OpenPGP card has an open specification, and open implementations available for Javacards. These cards, however, are only able to hold a single identity, itself comprising three 2048-bit RSA keys; one for decryption, signing, and authentication. The ability to securely use a large number of keys is therefore a clear advancement to the state-of-the-art, allowing a user to carry a single smartcard which provides access to a large number of identities.

A low-cost, scalable solution to the challenge of key management is contributed within this section. The solution allows a single smartcard to be used with an effectively-infinite number of separate identities. These identities (and their corresponding keys) are entirely orthogonal, and cannot be correlated or linked by a hypothetical omniscient third party with access to all public keys. This allows a user to protect as many keys as they desire, without encountering any limitation based on the smartcard's storage capabilities. This also means that cheaper (lower storage) smartcards can be purchased, on account of the constant memory requirement, rather than a linear memory requirement as more keys are added. The inherent limitations of storage on smartcards has previously been the topic of other work [255]. The proposed solution therefore allows a user to carry a single smartcard, capable of utilising any of an unlimited number of their secure keys to decrypt or prove their identity, rather than only being able to hold a limited number of keys. By way of comparison with the state-of-the-art, in the case of the OpenPGP card, the three RSA-2048 keys it can hold are realistically sufficient for only a single identity (one key for decryption, one for signing, and one for authentication) [256].

The proposed solution makes use of a regular smartcard, and uses an implementation of logic within the client application on a PC or other reader to interact with logic within the applet, and allow for the secure loading and offloading of keys. This is achieved by holding a pair of symmetric keys (one for encryption, the other for authentication) within the smartcard, using the regular properties of the smartcard to prevent them from being extracted or observed. These keys may then be used to decrypt incoming keys which were previously offloaded from the same smartcard. This removes the constraint of holding each key on the smartcard, while also ensuring that the offloaded keys are secured using standard symmetric encryption and ciphertext authentication, to avoid an attacker being able to use the offloaded keys in any environment other than on the smartcard used to generate and offload the key in question.

In addition to asymmetric keys capable of representing a user's identity through decryption and signing of data, the contributed implementation also allows for the secure offloading of symmetric AES keys, used either for confidentiality or authentication of data, where the data is not shared with other individuals, and thus symmetric cryptography is desired. This is designed to offer a means of protection for secrets, without exposing a single master key to the host computer. Rather, an offloaded key may be loaded, and used to decrypt a given file. It also increases the feasibility of implementing previous research based upon storing indentity-based attributes on smartcards, while reducing the cost of necessary storage on smartcards [257].

7.6.1 Overview of Solution

Within the proposed solution, the smartcard is designed for use in all cryptographic operations using long-term keys. In the case of the encryption and decryption of large quantities of data (such as user files), the smartcard will carry out the necessary cryptographic operations to recover a per-file encryption key to be used with a file. For asymmetrically encrypted data, the smartcard will decrypt an RSA ciphertext, and provide the host system with the per-message symmetric key. This ensures that a minimum of data is exposed to the computer at a time. Indeed, it reduces data available to the host computer to the minimum extent possible while still providing usable access to a system — in order to read an encrypted email, it is fundamentally necessary for the host device to have access to the plaintext of this email. By constraining the system's access to only the symmetric key needed to decrypt this one email and any associated attachments, the confidentiality of the remainder of messages is protected, even if the host system is an untrusted system, such as a public computer.

Likewise, where a user does not wish to share or receive data with others, and simply wishes to keep it confidential, the same principle is applied, and the smartcard will only reveal a per-file or message symmetric key to the host device. This ensures, once again, that a key with the lowest possible scope (covering only the entity being accessed) is made available to the host system.

While a system such as this would conventionally require large numbers of keys, and pose a complex key management challenge to users, this solution requires only a single smartcard, capable of protecting an unlimited number of asymmetric and symmetric keys, through the use of secure key offload. The offload process allows for the host device to retrieve an encrypted and authenticated representation of a symmetric or asymmetric key, without being able to use, or ascertain the identity of the key when offloaded. It may

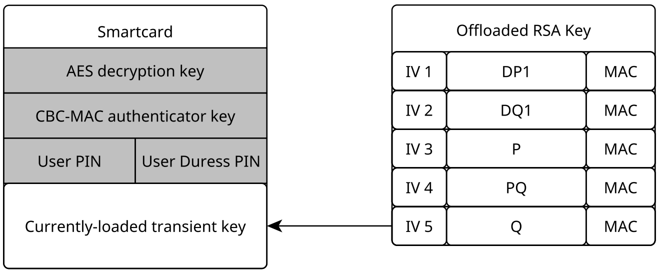


Figure 7.1
Architecture of key offload solution. Components in grey are held securely within the smartcard and cannot be exported or retrieved. The offloaded key may be loaded into the transient key slot of the smartcard, by loading each of the components of the key shown in the offloaded key.

later be loaded onto the smartcard for use. This solution has been designed with the assumption that offloaded key data is not held securely by the user, and that it may be accessed by an adversary or other untrustworthy third party. This assumption reduces the challenge of key storage to users, since keys may be held on cloud storage, and backups may be given to friends or colleagues, without exposing the user's keys. Indeed, keys may be made available publicly, without revealing any information about the key or its owner.

Figure 7.1 shows the architecture of keys used in the proposed solution — the smartcard stores various keys and PINs persistently, which are illustrated in grey within the smartcard. These PINs are used to validate the user attempting to use the card is permitted to do so. If user authentication is successful, the stored MAC and AES keys can be used to validate and decrypt offloaded keys respectively. An offloaded key component is shown as an example, being loaded into the transient key area of the smartcard. Each component of the offloaded key is encrypted individually by the card's AES key, with a unique random IV, and the overall key component is authenticated using the card's MAC key. The MAC of each component covers the full component, including the IV. The loaded transient key is then used to carry out cryptographic operations. To preserve the security of the internal card keys, the card's AES and MAC keys are not available for any use other than decryption of existing keys.

In order to remove the storage constraint of conventional smartcard solutions, the contributed approach was designed to securely offload the cryptographic keys from the smartcard, therefore no longer requiring them to be held within the smartcard. To ensure they remain secure, the keys themselves are offloaded in an encrpted format, and may be held on an external storage device. Since the keys are protected, they may even be stored on the internet, or otherwise distributed to multiple locations (offering simple access to the keys when required). Within the context of a decentralised storage network, it would be possible for the offloaded keys to be stored on the decentralised network itself, and retrieved by the user when attempting to use the keys.

CBC-MAC has been shown previously to be secure, provided the underlying block cipher is secure [258], which is currently believed to be the case for the AES block cipher (originally known as Rijndael [259]), given its selection as a NIST standard, its continued use in internet standards, along with its recognised resistance to viable cryptographic attack [260].

The proposed solution allows software on a user's computer (or mobile device) to interact directly with the card using APDUs (Application Protocol Data Units), as defined in ISO7816-4 [261]. Figure 7.2 illustrates the format of a typical host-to-smartcard APDU, and the components which form it. The `CLS` field is a single byte, indicating the class of the applet, which is used to identify which applet the incoming command is intended for. The `INS` field is a single byte indicating the instruction within the applet which is being invoked. The next two fields are a single byte each, and are `P1` and `P2`, which are two parameter fields for the instruction. These are used to pass a parameter or other extra information to the instruction being executed. The next field is the single byte `Lc`, which is used to indicate the length (in bytes) of the following data section. `Lc` may simply be set to zero if no data is required by the instruction. Finally, `Le` is a single byte to indicate the maximum response length expected by the host device. If no response data is expected, this field may be set to zero.

7.6.1.1 Key Lifecycle

Asymmetric identity key generation is carried out on the smartcard, initiated by the host device requesting the generation of a new key. Key generation will erase any asymmetric key held on the card, since keys are not stored on the card. Following generation of a key, it must be securely offloaded from the card, through the export process. The result of this is an offloaded key, containing the asymmetric key, in encrypted and authenticated form. For RSA keys, Chinese Remainder Theorem format private keys are used for offload, and each of the 5 Chinese Remainder Theorem key components (`DP1`, `DQ1`, `P`, `PQ` and `Q`) is exported. To prevent key components being incrrectly loaded (for example, loading `P` in place of `Q`, or replacing `PQ` with `P`), which would expose the implementation to key recovery attacks, only the correct key component may be loaded. The Key ID field is used to prevent this, as described in Section 7.6.4.

The offloaded form of the private key is provided to the host device, which should store the key appropriately, in a location where it can be retrieved in future. Since the offloaded key is encrypted using a key held only within the smartcard, exposure to a third party will not result in the compromise of this key component.

Following the offload of a private key, the host device may request the corresponding

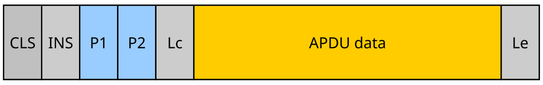


Figure 7.2
Generic Host-to-Smartcard APDU

public key component. For RSA, this is provided in the form of the public modulus, and the RSA exponent. Since keys are not held on the smartcard itself, and must be offloaded to ensure they are not lost, the smartcard applet was designed to prevent the retrieval of the public key until such a time as the key has been fully exported. This is to prevent incorrect use of the smartcard, resulting in a user no longer having access to their keys.

At an arbitrary point in the future, the host device may load a set of keys onto the smartcard, in order to allow for the use of an existing key. The offloaded key is transmitted to the smartcard, where it is validated and decrypted. The asymmetric key is then loaded and available for use on the smartcard.

7.6.2 Basic APDU Message Structure

This solution allows software on a user's computer (or mobile device) to interact directly with the card using APDUs (Application Protocol Data Units), as defined in ISO7816-4 [261]. Figure 7.2 illustrates the format of a typical host-to-smartcard APDU, and the components which form it. The `CLS` field is a single byte, indicating the class of the applet, which is used to identify which applet the incoming command is intended for. The `INS` field is a single byte indicating the instruction within the applet which is being invoked. The next two fields are a single byte each, and are `P1` and `P2`, which are two parameter fields for the instruction. These are used to pass a parameter or other extra information to the instruction being executed. The next field is the single byte `Lc`, which is used to indicate the length (in bytes) of the following data section. `Lc` may simply be set to zero if no data is required by the instruction. Finally, `Le` is a single byte to indicate the maximum response length expected by the host device. If no response data is expected, this field may be set to zero.

7.6.3 Secure Treatment of Keys

To protect the security of cryptographic keys, and ensure they are never made available to the host device, a robust set of security measures are put in place. These measures assume that the host device communicating with the smartcard is untrustworthy, or otherwise compromised, and therefore do not entrust any secure material to the host device. When keys are exported from the smartcard, they are protected through a number of precautions.

Firstly, a unique 128-bit initialisation vector (IV) is generated, using the smartcard's hardware random number generator, for each export operation carried out by the smartcard. This is to mitigate against the risks of re-use of IVs within the cipher-block-chaining (CBC) mode of AES. The key is then encrypted symmetrically using AES-256, with a key that is held on the smartcard. To authenticate the resulting ciphertext and ensure that an exported key which has been corrupted, modified, or tampered with is detected, the symmetric CBC-MAC authenticator is used, itself using a different key from the encryption operation.

Figure 7.3 illustrates the format of the resulting key, when exported from the smartcard.

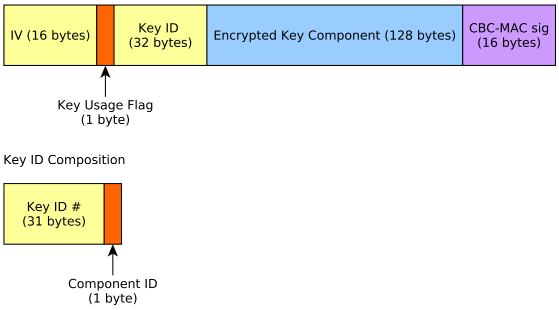


Figure 7.3
Encrypted key blob format

The resulting offloaded key may be held on storage which would normally be considered insecure, or otherwise lacking in the requisite confidentiality for the storage of cryptographic keys. This presents an advantage for users, since keys can now be safely held on a system where they could be reached by a third party, without fear of compromise.

7.6.4 Key Component Security

In order to prevent an attacker from carrying out a cut-and-paste attack, which could result in an attacker being able to swap key components and compromise offloaded keys, the key ID field is used as an additional security measure. As shown in Figure 7.3, the key ID field consists of two values; a randomly generated 31-byte value, to prevent key components from being swapped between different keys, and a 1-byte field containing the component ID of the key. Each key component has an identifier, corresponding to the field within the smartcard to which it may be assigned. During a loading attempt, the least significant byte of the key ID field is checked by the smartcard, to ensure that the component is being correctly loaded. This prevents the loading of, for example, a `P` component to the `PQ` field on the card.

7.7 Principles of Security

In order to ensure that good practice was followed, reference was made to Anderson and Needham's principles for robustness in public key cryptography [262].

7.7.1 Combination of Encryption and Authentication

In their principles, Anderson and Needham state that signing should always be carried out before encryption, in order to avoid providing proof to passive third parties of the authenticity of the ciphertext [262]. This is significant, since it is desired within the proposed solution for minimal information to be available to a third party with access to offloaded keys. Specifically, it is desired that no adversary may determine which smartcard may load a given offloaded key component, to preserve user deniability, as discussed in Section 7.10.1.

In our implementation, since authentication of the ciphertext is carried out symmetrically using a MAC, the signature cannot be verified without access to the card's symmetric key, which is held only on the smartcard and not exportable. The MAC therefore does not reveal any information to an adversary, and cannot be used to confirm the validity of a ciphertext without access to the authentication key held on the smartcard. This solution was implemented using this approach to ensure that offloaded keys may be verified prior to being decrypted and loaded, in the principle of defence-in-depth, to ensure that an invalid key is rejected as early as possible. This therefore reduces the risk of a maliciously generated key being used as a vector of attack against the smartcard.

Previous work has also considered the significance of verification within key-offload scenarios. For example, the key conjuring attack discussed by Clulow [253], highlighted the risks of allowing any arbitrary 8-byte sequence to be decrypted by a bound key storage device as a DES key, since it provided the opportunity to build a large set of keys for a parallel search to be carried out. This attack is mitigated through the proper enforcement of the MAC, thus authenticating keys being loaded to the unit. Since all offloaded keys supported by the unit are generated using the hardware random number generator on the smartcard, the risks they are generated insecurely externally is alleviated. Since all functionality using the onboard AES keys requires authentication to the card using the PIN before it may be carried out, it is therefore not possible for an unauthenticated attacker to generate new offloaded keys, or to attempt a chosen-ciphertext attack, due to the validation of keys.

7.7.2 Key Usage Enforcement

Per Anderson & Needham's principle 2, this solution is designed to enforce the proper separation of key usage. This is important when keys are exported, since contextual information which would normally be held within the smartcard, such as the variable name used to refer to a key, is lost when the key is offloaded. Therefore, when keys are stored permanently on the card, it is possible for the developer of the applet to ensure that different keys are used for signing and decryption. When key offload is carried out, it is necessary to also ensure that the same RSA key cannot be used both for decryption and signing. If both operations were permitted, an adversary could request the signature of a short ciphertext, and would receive the decrypted output, assuming that raw cipher operations . Since the key is offloaded in 5 components, it is necessary to ensure that each component is bound by the same usage constraints, to prevent unintentional or deliberate attempts to sign with a decryption key, and vice versa. The proposed implementation achieves this through the use of a key usage flag, which is contained within the MAC-authenticated key offload component.

At the point of key generation (which is carried out on the smartcard), a mandatory key-usage flag must be specified, either for signing or decryption. It is not possible to leave this field unspecified, nor is it possible to define a key which can be used both for signing and decryption, since the two are mutually exclusive. When loading an asymmetric key into the smartcard for use, the key usage flag is set using the bitwise-OR operation. This ensures that if blobs from different keys are loaded, or mixed, the key will not function in either mode. The key usage flag is cleared only upon generation of a new key, which erases the loaded key first, or upon manually erasing the loaded key to load a new one. Such strict enforcement ensures that it is not possible to carry out signing or decryption operations

In addition, to ensure that keys cannot have their usage flag modified after creation, the key usage flag is authenticated using the MAC of the offloaded key component. The smartcard will reject (and refuse to load) any key components which have a modified key usage flag.

7.8 Security Threat Model

The contributed solution is designed to allow for as wide a threat model as possible, in order to make it practical for use in a wide variety of situations. A *defence-in-depth* approach has been taken, where as many steps are taken as possible to ensure that confidential data is not revealed. It was assumed, and taken into consideration from the outset, that a user would store their key blobs in an insecure location, such as on cloud storage, or on a filesystem to which potentially untrusted software may have access, such as a standard computer or mobile device. Key blobs are therefore tamper-evident, on account of the embedded MAC for content authentication. In the event that a key blob is modified, the smartcard will immediately cease attempting to load it when the MAC verification fails. This also avoids the smartcard processing untrusted input, in case a weakness in the loading of key parameters was to be identified in a smartcard.

A potential compromise is considered as being a situation where any of the following occurs:

- the smartcard's internal AES key or CBC-MAC key is obtained or otherwise extracted from within the smartcard

- the smartcard's internal AES key or CBC-MAC key is altered, corrupted or otherwise changed

- the IV selected for encrypting an outgoing key blob is able to be chosen by an attacker

- unencrypted Chinese Remainder Theorem private key components of any RSA key handled by the device are able to be obtained from the smartcard

- an unauthenticated user is able to carry out a sign or decrypt operation using an RSA key handled by the device

- the key usage flag can be ignored or bypassed, allowing an RSA key to be used for both signing and decryption, or be exported with an incorrect key usage flag set

- a corrupted or otherwise modified key blob (including usage flag) is able to be loaded

- any internal state data is disclosed to the user unintentionally

- an external attacker may gather side information as to a user's identity, based on an encrypted key blob, other than the key usage flag value

- an external attacker may determine that a given smartcard has the ability to decrypt a given key blob

In contrast, the conventional security assumptions of a smartcard are that keys should be securely held within protected memory. Keys should only be accessible when used in an authorised manner, which may require authentication through a PIN or similar. The key on a smartcard would ordinarily be designed to be generated within the smartcard, and not exportable, thus guaranteeing that the secret key material does not leave the smartcard. Physical chip security design features are used to attempt to prevent access to the memory containing keys, and to provide properties of tamper resistance [263]. These properties are used within the proposed solution for the protection of the card-based MAC and AES keys, which are used as the underlying keys to decrypt offloaded keys. Therefore, if a given smartcard is sufficiently protected to prevent keys from being extracted by an adversary under conventional use-cases, our proposed solution will offer equivalent security to keys offloaded from the same card, since decrypting those keys requires the use of the AES key held only within the card.

7.8.1 Preventing Key Theft

In order to prevent the extraction of keys, appropriate JavaCard constructs, as per the API, are used for the storage of keys. The internal encryption key is held as a 256-bit `AESKey` object, and the internal MAC key is held as a 128-bit `AESKey`. Likewise, 2048-bit RSA keys are held (temporarily) within an `KeyPair` object, with appropriate private and public containers used for access to the components of the key.

The only time the AES encryption and MAC keys are written to is during the first initialisation of the card. During this point, their values are temporarily held in a byte array, whose content is erased on every applet selection and deselection, as well as prior to every future use, and which is defined as transient memory within the JavaCard API. After this point, and within regular APDU selection code, there is no write operation ever carried out on either key. No APDUs allow access to any key object or (in the case of RSA) unencrypted component. Additionally, at no point is any sensitive key material ever placed in the APDU output buffer, even temporarily. By doing this, the risk of fault injection attacks is reduced, where a subsequent instruction could be skipped, thus returning the (non-final) contents of the buffer.

7.8.2 IV Selection

Proper selection of initialisation vectors for AES is ensured, by directly requesting random data from the hardware random number generator of the smartcard at the point of use. This data is then used directly as the IV for the encryption operation. After each operation involving IV data, it is cleared, in order to prevent extraction of this data in future. Additionally, no encrypted data is ever returned without randomly generating an IV directly before carrying out the encryption operation. Following any other use of the IV array, such as when loading keys, its value is erased. Likewise, the array itself is transient, to ensure its state is not held between separate applet usage sessions. It is not possible to load an IV from the host device prior to an encryption operation, and after generating the random IV.

7.8.3 Protection of RSA Components

While it is necessary for encrypted key blobs to be exported from the smartcard, measures are taken to prevent the unencrypted key components from being retrieved. Firstly, at no point is any unprotected key data placed into the outgoing APDU buffer (as discussed in Section 7.8.1). Additionally, all temporary buffers used in the process of exporting the key are cleared, both at the end of the operation. They are also cleared at the start of all other operations, and are defined as transient so their values are not held between separate selections of the applet.

7.8.4 Preventing Sign or Decrypt Operations

In order to prevent an unauthorised user from decrypting data or generating signatures, the JavaCard `OwnerPIN` construct is used to protect the card and its state. Prior to carrying out signing or decryption operations, the verification status of the PIN is checked. This status is held within a transient variable, as per the JavaCard API documentation. Checks are also carried out for the failure first, in order to reduce the risk of a fault injection attack, which may attempt to skip the conditional jump after the verification instruction, and proceed with executing the code for a valid PIN being presented. Additionally, a limited number of PIN verification attempts are permitted, and management of this restriction (as well as changing the Owner PIN) is only possible after verification with the existing PIN.

It is recommended that in order to ensure the security of the Owner's PIN, a smartcard reader featuring a dedicated PIN pad be used, such that the Owner PIN is not disclosed to the host device, which is not necessarily trusted. On these smartcard readers, the PIN entry and validation process is carried out without any communication through the host device.

7.8.5 Enforcing Correct Key Usage

To prevent a signing key from being used to decrypt data, and vice versa (since signing is effectively the same crytographic operation as decrypting), each key blob is stored with an authenticated flag defining its use. This is set at the time of key generation, based on a user parameter. The key usage flag is implemented as two adjacent bits within a byte, and they are defined to be mutually exclusive. A request is always rejected if both bits are encountered as set or unset, or if any value other than the two permitted values is observed. When key blobs are loaded to the smartcard, it is necessary to first clear the existing key (which also clears the key usage flag to an invalid state of zero). Following this, the 5 key blobs should be loaded to the device in any order. As each key blob is loaded, the usage flag is logical OR'd with the existing key usage flag. If, upon attempting to use the key, the usage flag is not either of the permitted values (`USAGE_SIGN` or `USAGE_DECRYPT`), an attempt has been made to load different parts of the same key, and the operation is rejected. Similarly, if the key usage flag is incorrect for the operation being attempted, the operation will be terminated.

7.8.6 Preventing Corrupt or Modified Keys from Use

To prevent a modified or corrupted key blob from being loaded, each key is verified using a symmetric CBC-MAC authenticator as it is loaded. This authenticator covers the key usage flag, the initialisation vector, and the key data itself. If the MAC on a component fails, the component is not decrypted or loaded, and is instead discarded. This prevents the use of a modified key, or of a key whose usage flag has been altered to attempt to carry out the wrong operation (signing on a decryption key, or vice versa).

Additionally, the CBC-MAC is only used for the purpose of validating key blobs, and there is no interface exposed (or present on the smartcard) for the generation of the MAC of arbitrary data. This is to prevent a scenario where an attacker was able to coerce or trick a user into symmetrically (rather than asymmetrically) authenticating an arbitrary message, which would then form a valid MAC for a modified key blob that would be accepted by their smartcard.

7.8.7 Preventing Disclosure of Internal State Data

In light of the reduced memory environment on a JavaCard, it is desirable to re-use temporary memory where possible. While conventionally it would be bad practice to store confidential data in a buffer or array which also is used for non-confidential data, steps have been taken to prevent this exposing confidential information. Firstly, at no point is unencrypted key material ever placed in the outgoing APDU buffer. Secondly, all buffers are cleared at the end of every instruction, before the result is returned. Additionally, all buffers are cleared prior to an instruction being executed, to ensure that, even if a previous erasure were somehow overcome by fault injection, the buffers should still be erased prior to use.

7.8.8 Side Information from Key Blobs

Since, as stated above, users are presumed to store their key blobs on untrusted storage locations, such as cloud-based servers or devices running potentially untrustworthy software, it is necessary to ensure that a third party (such as the operator of a storage service) cannot establish any information as to the key contained within a given key blob. For this reason, key blobs are not stored alongside their public key. If multiple key blobs are to be stored in one location, they should be stored within a list, simply numbering each key sequentially, and identifying the component identifier of each of the 5 components of this key. This ensures that no side information is leaked as to the corresponding public key of a given private key.

The key blob, as described in Section 7.6.3, reveals a randomly-generated IV, the key usage flag (itself only indicating if a key is for decryption or signing), the AES-encrypted key blob, and a CBC-MAC as a symmetric authenticator to prevent tampering. The IV itself is directly generated from the hardware random number generator on the card, and the MAC is the output of the AES cipher in Cipher Block Chaining mode, which was demonstrated to be a pseudo-random function in [258]. The MAC itself simply covers the data already visible in the key blob, and thus its failure would not pose a risk of leaking unencrypted key data, since the MAC is carried out using an entirely separate key from the symmetric encryption.

Similarly, it is not possible to determine if a given smartcard can decrypt a given key blob, since the smartcard itself does not provide any identifier or means to determine which unit it is. While this could be a downside for a user with many such cards, the security model is designed such that this should be unnecessary, and a single smartcard may be safely used for access to various keys. Since Owner PIN authentication is carried out prior to even validating a key to be loaded, it is not possible for an unauthenticated attacker to determine if a given unattended smartcard may decrypt a given key blob.

7.8.9 Resistance to Attacks Identified against PKCS#11

Hertl has previously considered a number of attacks against the PKCS#11 standard, which contains provisions for the binding of keys to a cryptographic token [254]. The contributed solution advances the state-of-the-art by ensuring it is not affected by these attacks. This section shall consider these attacks, to demonstrate that the design of this solution considers each of these attacks, and takes steps to ensure they are not applicable.

The first attack identified by Hertl was the key separation attack, possible within a standards-compatible PKCS#11 implementation. This is mitigated in the proposed solution by enforcing and constraining the card to have a single key used for offloading of keys. This key cannot be used for any other cryptographic operation in our solution, and can only be used upon data already validated by the symmetric MAC authenticator. The output of this operation is also not exposed to the user, only to the underlying AES engine on the smartcard, therefore preventing the card's internal AES key from being used by the user to encrypt or decrypt arbitrary messages.

The expanded key separation attacks affect PKCS#11 by extending the key separation attack to not be based around the use of the same key. Once again, the proposed solution is not affected by this attack, because it is not possible to wrap keys with an arbitrary key. It is also not possible to use the decrypt operation with a key originating from any source other than an authenticated ciphertext. Additionally, it is not possible to use the same key to encrypt a key during offloading, because the offload key cannot be arbitrarily selected.

Key conjuring attacks, originally introduced by Cortier *et al.* [264] are not practical within the proposed solution, because all offloaded keys are authenticated using the symmetric authenticator key, and this key is never exposed to the user for use in cryptographic operations. No key will be decrypted or accepted if the authentication value is incorrect, thus preventing attackers from attempting to load crafted keys. In addition, it is necessary for the user to be authenticated to the smartcard by PIN before keys may be loaded, ensuring only authenticated attackers may attempt to load keys. The symmetric MAC contained within each exported key component prevents arbitrary keys from being loaded by authenticated attackers.

The key binding attack discusses the risks of permitting an attacker to load split keys in multiple operations, presenting a composite form of key which may be attacked [265], using techniques such as meet-in-the-middle. This attack is mitigated within this solution through the presence of the key identifer tag, generated randomly using the smartcard's hardware random number generator, and contained within each offloaded key component. As discussed in Section 7.6.4, this identifier also protects against the swapping of components of keys, which could permit a cut-and-paste attack to be carried out..

The weaker algorithm attack, where a key is wrapped using a weaker encryption algorithm which is weaker than the underlying wrapped key and algorithm, is not relevant within this proposed solution, as only AES-256 is used for key offloading. This ensures that the offloaded key is protected adequately, as no weaker algorithms are permitted for key offloading.

The downgrade attack, where a key can be wrapped then unwrapped with different constraints is not possible within the proposed solution, since metadata surrounding the permitted uses of keys has been designed to be persistent from the point of generation onwards. Keys are offloaded in a format incorporating a MAC-authenticated usage flag field, and this MAC is validated before the key may be loaded, thus ensuring that the key's usage is as originally defined at the point of generation, and cannot be modified after that point.

The private key modification attack is mitigated through the presence of the randomly generated key identifier, as discussed in Section 7.6.4, which contains a 31-byte randomly-generated persistent component to identify a key uniquely, and a key component identifier to prevent the swapping of key components during the loading of a key.

The trojan wrapped key attack is mitigated within this proposal through the symmetric authentication of all offloaded keys. This prevents tampering or generation of a key outwith the smartcard, since such a key will be rejected and no attempt will be made to decrypt or load it, unless a valid MAC is present on the key.

7.9 Full Implementation and Validation Against Existing Services

To demonstrate that the authentication system proposed and contributed in this chapter works as described, and can be used to carry out practical authentication against existing services, a number of integrations were created with existing services and authentication protocols. Where possible, widely used and popular systems have been selected, in order to maximise the potential number of users who may benefit from more secure storage of authentication secrets for strong authentication processes.

7.9.1 2-Factor Authentication Using HOTP/TOTP

Firstly, the TOTP/HOTP two-factor authentication standards, widely used by internet services including Google, Github, Dropbox, Lastpass, Facebook and others, were selected. HOTP [266] is an HMAC-based symmetric authenticator function, designed to produce secure, short numerical outputs which can be easily typed by a user who is logging in to a service. Each invocation of an HOTP token will produce a new login token, by incrementing a counter, used as an input to the HMAC function. A shared secret between the user's authentication device, often a smartphone running an application to generate codes, and the service provider, allows for entered codes to be validated by the server. TOTP [119] is a derivative of HOTP, to produce time-based authenticator for the special case where the HOTP counter's value is set to a representation of the current time. Use of HOTP or TOTP ensures that even if a user's session is compromised, such as through a keylogger, merely capturing a user's password will be insufficient to permit the attacker to log in as the target user in the future.

With both HOTP and TOTP being widely used [267], the ability to securely generate such tokens without storing them on a smartphone, where they may be vulnerable to exfiltration through exploits, presents a clear advantage for the security of a user's accounts. Previous work has implemented hardware-backed TOTP authenticators, although this has typically focused on storing a limited number of tokens on the security device. For the case of the Yubico implementation, two HOTP or TOTP keys may be held within the device [268], where each additional key requires the use of another "slot" on the device. This presents a clear trade-off for users wishing to store a large number of TOTP/HOTP keys on a security device, which could be resolved through the use of secure key offload.

An implementation of the HMAC-SHA1 algorithm was created, and validated against the test vectors from RFC2104 [269]. Using this implementation of HMAC-SHA1, HOTP and TOTP were implemented within a smartcard applet. To avoid complex and slow modular arithmetic operations being carried out on the smartcard to convert the resulting binary hash output into a decimal number, by representing the selected hash output modulo 10^6, this process was carried out within the client application.

To facilitate the use of a large number of HOTP or TOTP services, the secrets are wrapped to a card-specific key prior to being exported. As described previously, this offloaded key is constrained in its use, and the resulting output is then signed with the card's symmetric authenticator, to prevent tampering with offloaded keys. A function to import and wrap a new HOTP/TOTP secret is exposed as an APDU command, allowing new secrets to be enrolled with the card, and the encrypted, smartcard-bound offload key to be returned.

The outputs of the resulting implementation were successfully validated against the test vectors from both the HOTP and TOTP RFCs [266, 119], and also against Google's online TOTP validation system, by enrolling the provided secret into the smartcard. Testing was then carried out every 30 seconds for 20 repetitions, over a period of 10 minutes, since the TOTP code is updated every 30 seconds. Since the output of the TOTP algorithm is that from the SHA1 hash, used in HMAC mode, an error in the implementation would be highlighted within a small number of repetitions, due to the avalanche effect on the output of the hash function. These tests indicated that the implementation was correct, and compatible with other implementations in use.

7.9.2 Account Key Authentication Against Let's Encrypt

Let's Encrypt is a widely-trusted certificate authority, included by default within all major browsers and operating systems [270]. The Let's Encrypt model for certificate issuance is based around the ACME protocol, itself undergoing standardisation and review by IETF [271]. Key to the Let's Encrypt issuance process is the use of an *Account Key*. Rather than rely upon conventional username and password authentication to a CA web portal, where domain validation can be stored per-account, the ACME proposal allows for fully automated certificate issuance through an API, as implemented by Let's Encrypt. To request a certificate, a server must generate an account key (typically a standard RSA-2048 key), and lodge the public key with Let's Encrypt during a registration API call.

Following registration, all requests against the API must be signed by the private key corresponding to the account, thus allowing the origin of API operations to be validated against the original owner of the account. One major limitation of this proposal is that, due to the relatively short lifespans of Let's Encrypt SSL certificates (90 days), users are encouraged to use a client which holds the account key on the web server, using this key to request a new certificate when the old one approaches expiry [272].

In the event of a web server being compromised, however, the storage of the long-term account key presents a weakness, in that the adversary may use this key to gain access to existing domain validations which have been carried out in the past. With this account key potentially holding validations for multiple domains, the ability to hold this key securely, outwith any internet-connected system, would give clear security benefits, by preventing compromise of the computer from resulting in the compromise of the account private key. This is a risk which has been identified within the Let's Encrypt community, but with no clear solution to date [273].

Let's Encrypt uses the `RSA_SHA256_PKCS1` PKCS#1.5 signature scheme for account key operations, which is not commonly available on smartcards. Therefore, an implementation of this signature algorithm was created, following RFC3447 [274]. In the absence of formally defined test vectors within the RFC, the implementation was validated against the widely-used PyCrypto library's implementation of PKCS#1.5.

Given the lack of widespread support for many algorithms within JavaCard smartcards, as discussed by [275], it was necessary to create a full implementation of the RSA signing construction from the basic RSA decrypt operation, since many smartcards did not provide raw signing functionality without padding or hashing, and very few cards supported SHA256 digests within signatures [275]. Using the documentation for PKCS#1.5, a function to generate compatible signatures was created from first principles. This implementation has been made available [1]. A modified version of the acme-tiny project [276] was used to ensure that messages could be signed by the smartcard, without requiring the key to be held on, or accessed by, the user's computer. All signing operations are carried out by the smartcard, and the resulting signature is returned to the PC client, for transmission to the Let's Encrypt server as part of the certificate request protocol.

The overall solution was tested against the Let's Encrypt boulder CA, using the publicly available development and testing API. A variety of requests necessary to validate a domain and request certificate issuance were tested against the Let's Encrypt API for 10 different private account keys, with no validation failures occurring.

7.10 Attack Countermeasures

In common with other smartcard-based solutions, this solution is based on the assumption of the security of the smartcard used to hold the cryptographic keys and carry out the operations. As discussed in [277], a smartcard is designed to be secure hardware, with tamper-resistance, and a restricted interface to ensure that only acceptable commands may be carried out. The assumption is also made that the user of the smartcard is the authorised user, through the verification of a PIN or other authentication process.

7.10.1 Mitigations Against Invasive Attacks

A wide variety of physical attacks have been attempted against smartcards, including simple [278] and differential power analysis [279] and fault attacks [280]. Firstly, to mitigate against fault attacks, which could be used to skip an instruction from being executed [281], all conditional checks (such as if statements) are designed to check for the negative. This means that a check for validity would instead be implemented as a check for invalidity first. This means that, in the presence of a successful fault injection attack, the condition may be skipped, and the failure code would be reached, rather than the success code [282].

In order to make power analysis more difficult, random *noise* is introduced to the process, by adding some extra instructions which generate random data, and store it within temporary arrays. Then, in order to introduce a second order of noise (and thus variable timing), further random data is generated, the quantity of which is based upon the previously generated random data. The purpose of this variation is to prevent static timing analysis taking place (such that an attacker could know validation of the duress PIN would take longer than a regular PIN). Additionally, as a second precaution, the

[1] https://github.com/greigdp/Javacard-ALG_RSA_SHA256_PKCS1

process of PIN validation has been designed such that, upon successful validation of either PIN, the same quantity of random data is generated, and an AES key is overwritten. When the regular PIN is entered, a dummy AES key is overwritten (which is itself located adjacent to the real key), and when the duress PIN is entered, the regular AES key is overwritten. This forms an implementation of the hiding technique discussed in [278, Chapter 7], where it was observed that "only a small number of scientific publications [focus] on concrete implementations of hiding countermeasures."

Finally, to add further to the variation of the operation of the PIN validation, the order in which the PINs are validated varies, depending on some of the second-order randomly generated data. This was implemented on account of the fact that a key security concept is to avoid branching based on sensitive data, since differential power analysis may reveal which branch condition was taken, based on whether a new code offset had to be calculated or not. By altering the order of verification based on random data, this again makes it more difficult for an attacker to determine if a duress PIN was entered.

A regular PIN entry operation will also generate an equivalent volume of random data from the hardware random number generator, and set the value of a dummy AES key to this newly-generated random data. Therefore, the difference between a valid PIN being entered, and a duress PIN being entered, is only the segment of memory to which the random number generator's output is written. This ensures that, even with differential power analysis taking place, it should not be possible to distinguish between a legitimate PIN being entered, and a duress PIN being entered (thus erasing the user's AES key and preventing the smartcard from being used to decrypt their keys in future).

7.10.2 Human Factors Including User Duress

The facility has also been added for a user under duress to force an erasure and regeneration of the internal AES encryption and authentication keys. This occurs during a login attempt, in a manner which does not reveal that the duress process has been carried out. The intention is that a user under duress can provide (or enter) a PIN which is accepted by the smartcard, and which irretrievably replaces the internal keys with new ones that are unable to decrypt previous key blobs. To prevent an adversary from knowing that this feature has been invoked (which may make it difficult for a user to deny they held the keys they are being forced to reveal), this feature has been designed such that, even under the observation of a skilled adversary (carrying out power monitoring of the smartcard), it should remain difficult to observe. This duress PIN will therefore enable usage of the smartcard, as though the regular PIN had been entered, although it will refuse to accept offloaded keys, in exactly the same manner as another smartcard would react when presented with keys generated from another unit. This feature is designed such that, in the absence of a user also being found with their own key components, it is not possible to determine if a given offloaded key was used by the smartcard, which would indicate that the user had invoked the duress functionality.

7.10.3 Verification of Duress Functionality

In order to demonstrate that the duress protection is resistant to timing analysis, an experimental setup was constructed in the best-case scenario for an adversary. In this scenario, the attacker was assumed to have direct access to the smartcard, and the ability to send PIN verification messages directly from a smartcard reader. The attacker is therefore able to measure the exact time between sending the message, and receiving the response. This was identified as a realistic attack vector, since modified client-side software on the computer would be able to carry out highly accurate timing, by measuring the response time of the smartcard under different operations.

Given the above mitigations presented, it is hypothesised that it should not be possible to discern between a valid PIN entry or a duress PIN entry, by observing the processing time. The JavaCard OwnerPIN construct was used throughout for PIN validation, and therefore safe, constant-time comparisons are used for PIN verification.

To validate this, 1000 correct PINs were supplied, along with 1000 duress PINs. Note that an invalid PIN can be detected by an adversary, since the user will be notified of an incorrect PIN being used. The scenario requiring deniability is the invocation of the duress PIN by the user, in order to allow the user to invoke this feature without alerting or drawing suspicion to their actions.

Figure 7.4 shows the result of comparing the time to process 1000 correct and duress PINs. The mean time to process an owner PIN was 18.206 ms with a standard deviation of 0.81 ms, and the mean time to process a duress PIN was 18.205 ms, with a standard deviation of 0.80 ms. There is no clear distinction between the owner PIN and duress PIN being invoked, indicating that knowledge of a single PIN entry event, which would be sufficient for the user to erase their keys using the duress key, cannot be distinguished from timing of the PIN validation process. This was confirmed by correlating the two distributions, resulting in a correlation coefficient of 0.9989 across 1000 samples.

7.11 PIN Entry Security

To prevent the security of the overall solution from being compromised through the user being required to enter their PIN on their regular computer keyboard, where it may be captured by a keylogger or similar, the solution presented features support for

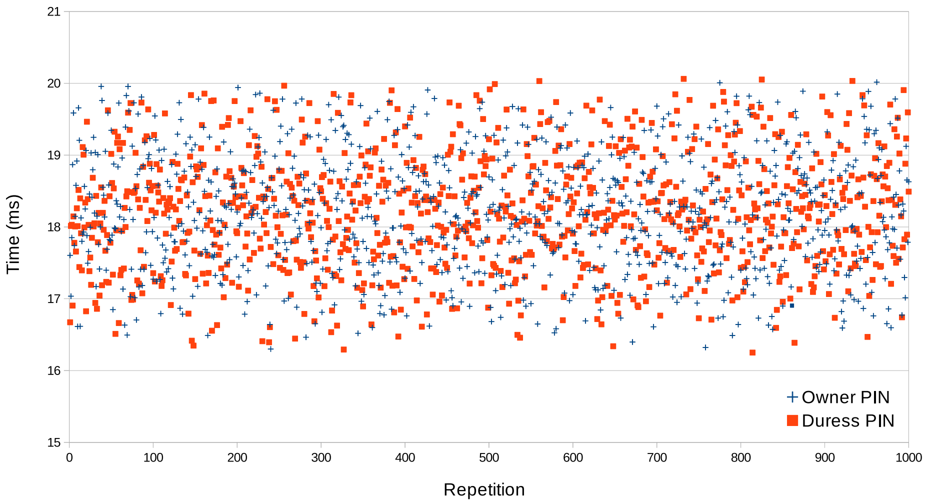


Figure 7.4
Time taken to validate 1000 valid owner PINs, and 1000 duress PINs.

smartcard readers featuring secure PIN entry capabilities, commonly referred to as PIN pads. The `ScardControl` interface [2] of the PCSC-Lite driver is used to interact with PIN readers.

Both a Gemalto IDBridge CT710, and a Cherry Smart Board XX44 were used, in order to ensure both PIN pads with and without screens would be usable. A driver to interact with the PCSC-Lite APIs was created in Go, and used to issue PIN verification requests, as well as to initiate PIN change operations. This allows for all operations requiring a user's PIN to be carried out entirely upon the smartcard reader.

The APDUs used to initiate a PIN verification and PIN change are passed to the reader, along with parameters to the `FEATURE_VERIFY_PIN_DIRECT` or `FEATURE_MODIFY_PIN_DIRECT` IOCTLs, as appropriate. These parameters indicate the position within the APDU at which the user's PIN should be placed, and how it should be aligned (left or right). By using ASCII representations of characters, the same PIN may be entered on a device without a secure PIN pad, simply using the regular numeric keypad. Variable length PINs are supported, provided they are within the length range allowed by the reader, since left-aligning the PIN ensures it can be validated correctly by the smartcard.

7.12 Performance and Discussion

The primary means of measuring the performance of this solution is through the profiling of the time taken to carry out various cryptographic operations on the smartcard, in comparison with the time taken to carry out the same operations on the untrusted

[2] https://pcsclite.alioth.debian.org/api/group__API.html

host device. For the purpose of this comparison, the performance of key generation, offloading, public key retrieval, message signing, and message decryption are considered.

In order for secure offload of keys to be a practical solution for users, an acceptable level of performance must be achievable by application software which is accessing the smartcard. To profile the performance, a series of performance tests were carried out on the smartcard applet, using an application designed to automate the repeated use of the smartcard for various operations. The mean time to generate a new RSA key, retrieve each component of the key, load these components in future, retrieve the public key, sign a message, and decrypt a message are considered.

All performance measurements were carried out using a physical smartcard, with the applet built against the JavaCard 2.2.2 API. Applets were compiled using the Oracle JavaCard JDK, tbhrough the Eclipse JCIDE plugin. The smartcard used was a JavaCOS A40, which is a CC EAL5+ verified smartcard, based on the Infineon SLE77 platform.

7.12.1 Key Generation

The process of generating an RSA key is carried out using the smartcard's integrated secure random source, and requires the generation of two very large numbers, which should be prime. This process requires firstly the generation of the large numbers, and secondly the verification of the likelihood of primality of the numbers selected. For this reason, the time required to generate a new RSA keypair may vary significantly, depending on the number of attempts needed to generate suitable candidate primes. This is similar to the process used on a desktop computer, where the time taken was found to vary from 31ms to 531ms in 20 repetitions, with a mean of 167ms. On the smartcard, key generation performance is inherently slower, with a mean time of 7.4 seconds required to generate a key, across 20 repetitions.

While key generation is therefore lower performance on the smartcard than on a desktop computer, this is an operation likely to be carried out relatively infrequently, at the point of creating a new identity or registration with a service. That a key was able to be typically generated in less time than it would take for a user to read a few instructions on a web page indicates that a mean key generation time of 7 seconds should not prove problematic for usability.

7.12.2 Public Key Retrieval

Similarly to the process of secure key offload, the process of retrieving the public key from the smartcard was timed over a series of 20 repetitions for different keys. It should be noted that this process only retrieved the modulus, on account of the exponent being a fixed constant value of 65537, thus not requiring retrieval on each use.

The mean time to retrieve the public key was measured as 40.1ms, with a standard deviation of 0.067ms, highlighting that retrieving or confirming the key present on the card at any given time does not require a significant period of time.

7.12.3 Signature Generation

The performance of signature generation on the smartcard was found to vary with the length of the message being signed, on account of a longer message requiring more bytes to be transferred to the smartcard over the physical link. On account of the equivalence of signing and decryption in RSA, signatures should always be generated of the cryptographic hash of a message, as discussed in [262]. This prevents a malicious party from requesting a signature of an intercepted message they wish to have decrypted. To mitigate this, as discussed in Section 7.7, this solution enforces separate keys for signing and encryption. For this reason, it is possible to carry out hashing of the message on the client computer, since even a compromised host device may not decrypt a message without user consent by requesting a signature. This also allows the direct signing of a longer message than 255 bytes to be carried out, without exceeding the capacity of a single APDU, and without requiring a reader which supports extended APDUs be used. Note that internally, the JavaCard available APIs specify that the SHA-1 digest is used on the message. This highlights the importance of smartcards being available using more recent cryptographic primitives, as highlighted by [275].

Figure 7.5 shows the time required to transmit a message of a given length in bytes to the smartcard, have a signature generated, and receive the RSA signature of the message. Signature generation performance was linear with respect to the length of the message to be signed. A signature of 1 byte of data took 5.5ms, and signature of 128 bytes took 22.0ms. This shows that signing operations, to authenticate requests, or otherwise prove identity, are able to be carried out in a very short period of time.

7.12.4 Message Decryption

The process of decrypting a message involves the loading of a 256-byte ciphertext into the smartcard, the decryption of the ciphertext using the loaded private key, and the returning of the plain-text message. Figure 7.6 shows the performance recorded On account of the varying length of the plaintext, and thus the quantity of data to be returned to the host device from the smartcard, the performance of a decrypt operation varied slightly with the plaintext length.

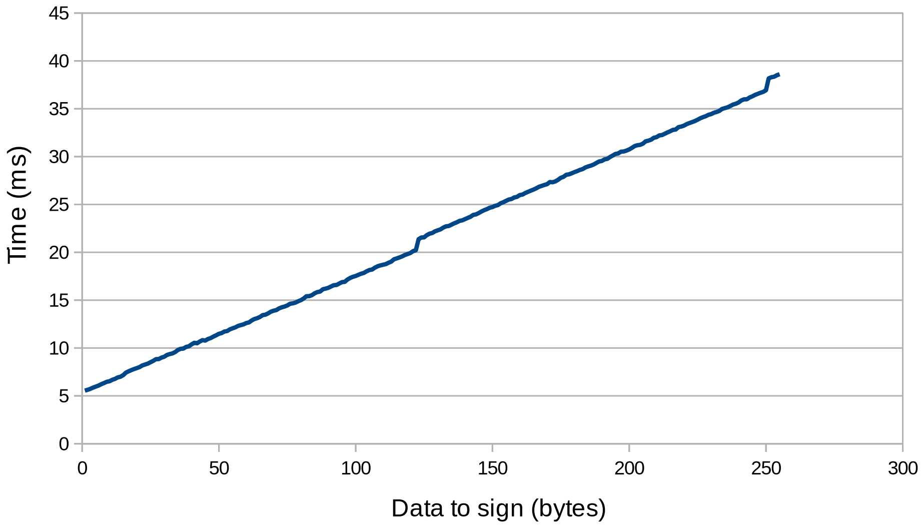


Figure 7.5
Signature generation performance for messages of lengths from 1 byte to 255 bytes

This performance indicates that it is practical for a decryption operation to be carried out by an asymmetric key entirely on the smartcard, to reveal either a short message, or a symmetric key used for a longer ciphertext, thus preventing the long-term asymmetric key from bbeing exposed to the host computer during the process.

7.12.5 AES Performance

In addition to tests carried out against the asymmetric identity key functionality, the performance of the symmetric key protocol was also evaluated. In comparison with RSA, and as is reasonable to expect, performance of the symmetric AES operations exceeded that of the asymmetric operations.

Across 2220 cycles of testing, generation of an AES key was found to take a mean of 20.3ms, with a standard deviation of 0.19ms. Likewise, retrieval of the protected (signed and encrypted) key took a mean of 28.5ms, with a standard deviation of 0.18ms. The process of clearing the loaded AES key took 15.1ms, with a standard deviation of 0.12ms. Finally, the process of loading an AES key into the smartcard again for later use took a mean of 44.1ms, with a standard deviation of 0.27ms.

The process of retrieval and loading of the key took longer, on account of the generation of the CBC-MAC across the encrypted key for export, and the validation of this MAC prior to decryption of a key being loaded.

Random plaintexts of various lengths from 1 byte to 220 bytes, which was the maximum ciphertext able to be output by the smartcard, were then encrypted and decrypted

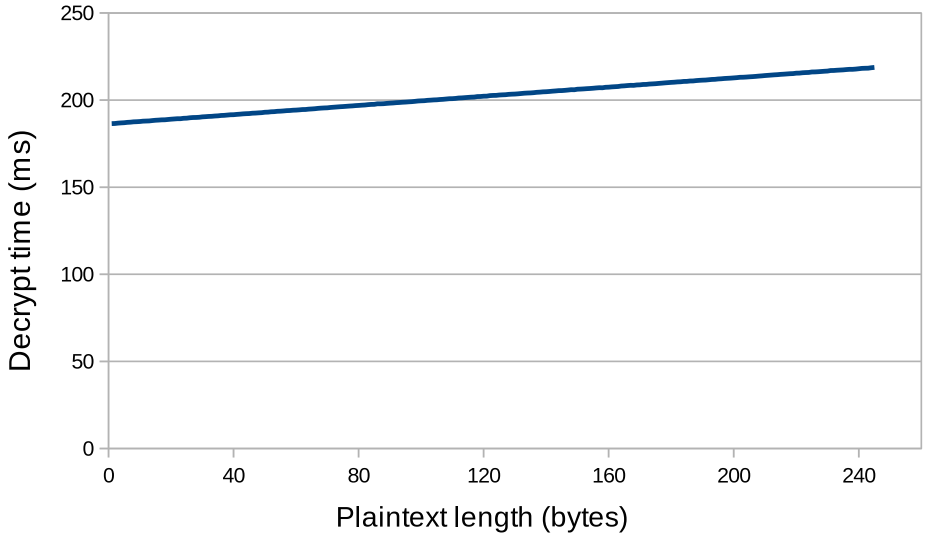


Figure 7.6
RSA Decryption generation performance for plaintexts of lengths from 1 byte to 255 bytes which were encrypted into a ciphertext output of 256 bytes

by the smartcard. The maximum permissible plaintext length was 220 bytes, on account of the overhead of a random initialisation vector, and PKCS#7 block-padding being applied to the plaintext prior to encryption. The performance of encryption is shown in Figure 7.7, and the performance of decryption is shown in Figure 7.8.

Of note is the stepping observed in the performance of encryption and decryption, on account of the AES cipher being block-based, and these steps corresponding to 16-byte increments in plaintext length.

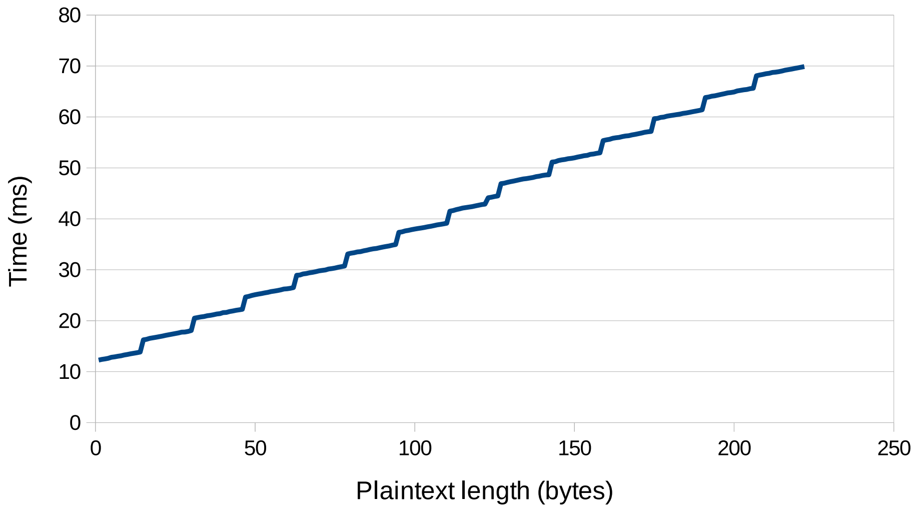


Figure 7.7
AES encryption performance for messages of lengths from 1 byte to 255 bytes

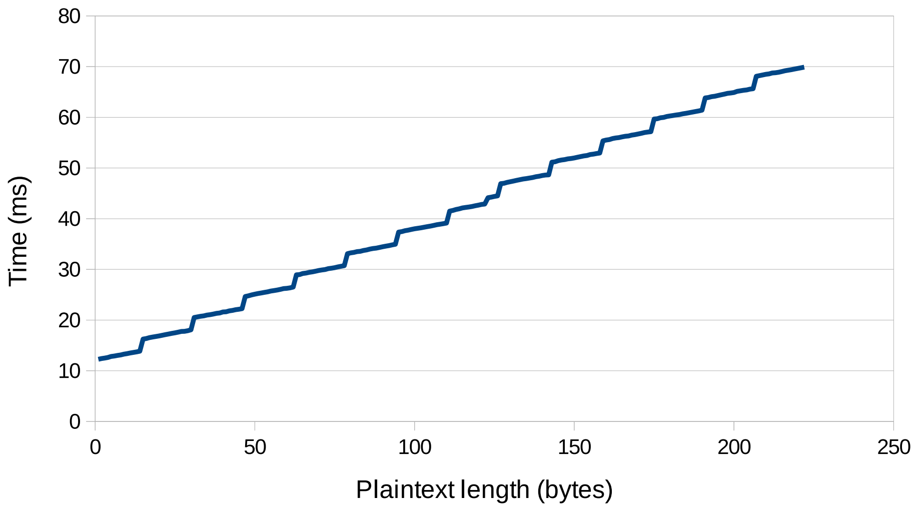


Figure 7.8
AES decryption performance for plaintexts of lengths from 1 byte to 255 bytes, which were encrypted by the card and subsequently decrypted

7.12.6 Identification of Performance Overhead

In order to establish the performance overhead of the proposed solution, compared with the state-of-the-art; namely to simply store a key on the smartcard and use the key for all cryptographic operations directly, the 2 areas of the proposal resulting in overhead are considered specifically. The only 2 changes introduced by the proposal, over the standard cryptographic operations of the smartcard, were found to be through the secure offloading of key components, and the re-loading of these components for later use.

7.12.6.1 Secure Key Offload

The key offload process is used following key generation, to retrieve a secured copy of an identity key, and hold it elsewhere for later use. There are 5 key components retrieved from the smartcard, as described in Section 7.6.1.1. To measure the performance of these operations, a series of keys were automatically generated and each component was retrieved from the smartcard. This process was repeated 20 times.

The time to retrieve each key component, including the time taken for the smartcard to encrypt it and create the CBC-MAC symmetric signature, was found to be constant, with a mean of 37.4ms, and a standard deviation of only 0.4ms. The overall time to retrieve all components of an identity key was therefore found to be around 186ms, on average. This is the only overhead at generation-time, introduced through the key offload mechanism proposed here.

7.12.6.2 Loading a Previously Offloaded Key

The only other overhead introduced through the proposed key offload technique is experienced when loading an existing key onto the smartcard. This must take place before the key may be used for cryptographic operations. The process of loading the components of a key consists of X steps; firstly, the smartcard is issued a clear key operation, to ensure the key usage flags are reset, and currently-loaded key components are erased. Secondly, the necessary key components are sent to the smartcard. Thirdly, the integrity of each component is verified through the symmetric authenticator. Finally, the key usage flag is updated for the now-loaded key, and the next key may be loaded. Once all keys are loaded, the loaded key may be used for cryptographic operations.

The clear of any existing loaded key was found to have minimal overhead, with a mean of 6.1ms required to clear any loaded keys, with a standard deviation of 0.48ms. The process of loading the 5 components of an offloaded RSA key was found to take an average of 187.5ms, across 20 repetitions, with a new key generated, exported, and then re-loaded to the smartcard. Again, there was minimal variation, with the standard deviation across a full load of 5 key components found to be 1.3ms.

As a result it is clear that for scenarios where a user is not rapidly switching between keys, the performance presented by the proposed solution is more than satisfactory. Indeed, since the overhead is only experienced during a key-load or offload event, which take place only during set-up of the card for a session, or new key generation respectively, the performance for cryptographic operations is unaffected. This was verified by comparing performance of the proposed offload-based applet with a standard applet using fixed keys. This is as expected, since the base cryptographic implementations of signing and decryption are unmodified, with the offload implementation containing these as a subset of its functionality.

7.13 Conclusions

In comparison with the state-of-the-art, where non-ephemeral keys are held on regular computers, a means of securely using cryptographic keys held on a smartcard has been presented, without the usual constraint on the number of keys capable of being held on the card. This process of secure key offloading allows for the use of low-cost storage (in comparison to the high-cost secure storage of a dedicated secure smartcard) to hold keys, including on storage which is potentially untrustworthy, or operated by a third party.

By holding cryptographic keys outwith the regular computer, and providing a single-purpose interface through the smartcard interface for cryptographic operations, it is possible to securely carry out operations using these keys, without exposing the keys to the host device. This allows for secure usage of keys on untrusted devices (such as a computer in a public place), without the risk of exposure of the long-term cryptographic keys used to form the user's identity.

Since the number of keypairs able to be used by the smartcard is not constrained by the smartcard's storage capacity, it is entirely practical for a user to generate a separate identity key for every service they use, or entity they interact with. This enhances their privacy, as described previously, and helps to protect their keys against compromise. Given the limited capacity of commercially available smartcards, this is a barrier to their adoption in user-centric deployments. While closed deployments, where the smartcard is managed by an employer and used only to authenticate to a series of closed services, are practical with current smartcards, this work presents a practical implementation of an open, user-controlled smartcard keystore, where a user may generate their own keys for identities, and manage these identities themselves, without needing to understand the specific process. Practical applications have been demonstrated, with Let's Encrypt account key authentication being implemented on the smartcard, and an implementation of smartcard-based TOTP two-factor authentication being created.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

Recent years have shown the scale of the challenges facing people wishing to hold their data secure, and preserve their right to privacy. Companies being entrusted with users' data have suffered from serious data breaches, through to rogue employees releasing sensitive data, through to vulnerabilities in software allowing unauthorised access to private data.

Even in an ideal scenario where all these problems could be resolved, there still remain a number of problems — the vast majority of today's services are designed around a model that trusts the service provider entirely with a user's data. There is potential for companies to spring up, in order to abuse this model of trust to gain access to user data. For example, many services have appeared, which seek to gather bills and financial statements, and store them within cloud storage services. All parties involved in this process could, if malicious or compromised, gain access to all of the data in question.

A stronger, more secure, and more privacy preserving way to design these services is to use a so-called zero-trust model, where service providers are not considered to be trusted, or required to be trusted for users to gain access to their data securely, and carry out their business. This presents major benefits for users' security and privacy — their data is confidential, even if the service provider were bought over or acts unethically, and they have full control over who, if anyone, may access their own personal data. This kind of effective control empowers users to exercise their right to privacy, and allows them to select who may have access to what information, and under what circumstances.

While this model helps to address the problem of trusting service providers, which has been shown to be a concern in this thesis, it does not resolve the challenge of ensuring availability of services, and ensuring users retain utility, even when a service provider ceases to wish to offer a service. There have been a number of high-profile cases where a service provider has ceased to provide a service, and users have been left with no clear migration path, or way to use their data. This presents a major risk for users, in that they may have to invest time or money into identifying a suitable alternative service, and migrating to it, without any clear reassurance this will not happen again in the future.

A decentralised approach to creation of services has been explored in this thesis, which aims to resolve these concerns of availability and longevity. By ensuring that utility and functionality cannot be lost in the event of the failure of the developer or operator of a service, a resilient infrastructure which will out-last individual services helps to ensure that users remain in control of their data, rather than having their data under the control of service operators.

Nonetheless, even in a decentralised services model, there remain important considerations around security, specifically around authentication and identification of users, and ensuring that only authorised users carry out operations. Additionally, the security of the client device used to access services remains a consideration, and the risks of overly-centralised approaches to the setup process of Android devices has been considered, with weaknesses identified and demonstrated on these devices, allowing Google, or other attackers, to violate the Android security model and compromise data held on devices. The risks of poor implementations of hardware security on low-cost devices is also significant, especially for users in the developing world, where their devices may be used by multiple people, and may be the only device used in their family for banking and other services. Finally, the risks of improper implementation of widely available and used security and encryption software has demonstrated the risks of using unaudited, closed-source software claiming to offer security through the use of encryption. Major flaws in implementations were found in software used by millions of users, including static keys used across all installations of an app, through to AES used in counter mode with static initialisation parameters, through to trivial obfuscation techniques and home-grown ciphers. All of these highlighted the risks of using software violating Kerckhoff's principle by failing to open their implementation to scrutiny.

Even with decentralised services, and a zero-trust model for service provision, there remains the challenge of how to securely authenticate users, since most current authentication processes rely on the service provider being trustworthy and carrying out authentication correctly, with the potential for complete account take-overs in the event of social engineering or weaknesses in implementation being shown. While passwords remain the current means of authentication for most services, a number of alternatives exist. The fundamental weaknesses of implementations and the concept of biometric authentication have been explored through an analysis of fingerprint authentication implementations, and a number of practical concerns surrounding this indicate that biometric authentication does not offer a solution to the problem of secure user authentication.

Combining this with the risks and challenges of a secure client implementation, as discussed above, an implementation of secure identity and key management on smartcards has been devised, and demonstrated, which allows for users to hold an unconstrained number of decentralised identity keys securely on a single smartcard. These keys may be used to encrypt and sign data, without exposing the underlying keys to any client device, significantly reducing the attack surface exposed towards keying material and other long term keys.

One concern surrounding decentralised and zero-trust services is that it can be difficult for users to re-key, particularly in current implementations of such services, in the event that the user believes their keys may have been accessed or stolen by another party, or even if they wish to change their authentication password for the network, if they are not using smartcards or similar for identity management. In such a scenario, the transferable ownership of data within a decentralised network was introduced in order to provide a means for a user to re-key their data, and change it to be owned by a different identity, allowing it to be transferred to a a new network identity if the first was feared compromised, or if a user simply wanted to separate out different data.

To show that decentralised and zero-trust services are viable, such a solution to the challenge of secure contact discovery has been demonstrated and evaluated from a security perspective, to show that it is possible for two mutual contacts to discover each other amongst service users, without disclosing any details about either party's identity to the service operator, and without allowing the service operator to monitor access patterns to establish relationships between different users. It is shown that this can be implemented entirely using key-value pairs of data, and that therefore this presents a viable decentralised solution to the ongoing unsolved problem. This may be used in any scenario where two users, that each know the other's identifier such as a phone number or email address, wish to determine if the other uses a given service or protocol, without revealing the identity of themselves or the other party to the service. This helps to comply with legal requirements in Europe which require consent of data subjects for their data to be processed; something which is not currently happening in many state-of-the-art implementations, where users disclose their contacts to the server (without their explicit consent) to determine if any of them use the service.

8.1.1 Review of Key Contributions

In exploring the above, a number of contributions have been made to the field, which address the research questions posed in Chapter 1.

First, the question of how vulnerable a decentralised storage network is to large numbers of malicious users joining the network was addressed in Chapter 3, with a review of the protocol used to establish how nodes and managers are allocated. This was published in [A1]. The algorithm for determining proximity of nodes was analysed, to produce a prediction as to the difficulty of compromising nodes within the network. This was demonstrated to be a feasible and practical attack through the generation of a large number of identities in a highly efficient manner.

The question of how to secure a decentralised network against large numbers of rogue users was also addressed in Chapter 3, with the contribution of a high-performance and efficient proof-of-storage protocol.

This was then built upon in Chapter 4 to address the third research question, around whether secure agreements can be reached amongst untrusting parties in a decentralised network. The secure decentralised digital contracts solution contributed shows that this can be achieved, in an efficient way, and was published in [A3], within the context of building contracts to allow for the purchase or sale of storage on the network.

Whether smartphones and other mobile devices could feasibly connect and participate in a decentralised network was also explored in Chapter 4, since a useful secure storage network requires convenient access, as well as advanced capabilities such as buying and selling storage. A hybrid relay-based solution was contributed, showing that it was feasible for mobile devices to use a relatively-untrusted relay server to act on their behalf on the network, while avoiding disclosing plaintext contents to the relay server, protecting their data.

Next, given that users may wish to access their storage from mobile devices, the security of these was considered, to address the research question posed around whether users are at risk as a result of such software, and how this could be mitigated. This was explored in Chapter 5, which contributed a security evaluation of the Google components of the Android platform, which has been published in [A4]. Further work on the security of specific Android hardware devices was published in [A14]. The security of software implementing encryption on Android was also considered, to address the software side of the research question, since pivotal to a decentralised storage network is the local encryption of user data, to avoid exposing it to third parties. This was published in [A21].

The question of how to discover other users and content within a decentralised network, without centralised servers for indexing and finding common contacts, was addressed in Chapter 6. An approach was contributed, which solves this problem in an efficient manner, scaling in complexity on the client side by the size of a user's contact list. The resulting protocol offers measurable protection against attackers attempting to ascertain users' social graphs, and this is quantified.

The final two research questions, around how users can protect and secure their own data when it is fundamentally exposed to others, and around allowing users to have multiple secure identities, such as to separate work and personal files, were addressed together in Chapter 7. Here, an approach to the storage of long-term identity keys on smartcards was contributed, such that the number of identities is not constrained by the capacity of the smartcard.

8.2 Future Work

A number of areas of future work have been identified in the course of this thesis. This section details these, and some considerations which may be of interest to those keen to further explore the fields of work discussed in previous chapters.

8.2.1 Decentralised Networks

This thesis has primarily focused on a small number of specific security considerations of the MaidSafe network. Where possible, these have been approached in a manner which is as generic as possible, while still remaining accurate and useful. There would be benefit in future work exploring the security of other decentralised networks — much of the existing literature has focused on distributed hash tables, rather than on secure implementations of services upon them, and it is likely that there remain weaknesses in other implementations, perhaps similar to those identified here. Specifically, Chapter 3 showed that the XOR metric in itself does not provide a strong guarantee as to the difficulty of creating a chosen cluster of conspiring nodes on a network. If other services or implementations make similar assumptions as to the distribution of members of their networks, they may be vulnerable to similar attacks to those showed here.

8.2.2 Identity Management

One of the fundamental assumptions made in the identity implementation of the MaidSafe network was that user private keys could be derived from a secret credential, such as a password. Good practice is to ensure that all passwords are regularly changed, although changing a password in such a scenario would prove problematic — since the password is used as the input to a key derivation function, changing the password will result in a new key being derived.

During the login process, there is a need to ensure that appropriate key derivation function parameters are selected. For the example of scrypt, there are 3 parameters to select. In the event that these are changed, it will not be possible to derive the same key as previously, from the same credential being input. This therefore means that there is a requirement to correctly identify the correct parameters to use, in a secure manner. If a new user to the network can be "tricked" into using low parameters, this will make it easier for a malicious entity to attempt to brute-force their credential. Nonetheless, it is also necessary that an existing user logging in from a new device many gain use the same parameters as previously. One naive approach would be to store the login parameters on the decentralised network in plaintext, although this would also be visible to third parties wishing to attempt to find users with weak key derivation parameters. Future work could focus on addressing this challenge, and attempt to securely ascertain the correct parameters to use for an existing user logging in. Solving this would make it easier to allow a user to select arbitrary parameters at login, perhaps based on the processing speed of their computer.

Another area of potential interest for future work would be on secure implementations of newer, more modern cryptographic constructs on smartcards. While this work focused on the design of a secure system to use smartcard-based keys for multiple identities at once, which can be disposed of after use, the performance of RSA keys is a limiting factor, especially on embedded systems. A correct and secure implementation of asymmetric constructs based around Edwards curves (such as Ed25519) may offer better performance and security properties for a smartcard. In addition, these use much shorter public keys, and have safe, well-reviewed high level implementations allowing for their use in authenticated encryption and deterministic signature generation.

8.2.3 Client Security

In investigating client and endpoint security factors, it is clear that real-world implementations of encryption software often leave much to be desired. The most obvious piece of future work would be to extend an existing automatic application scanner to attempt to identify use of weak cryptographic constructs. Similar work has been carried out to attempt to find applications which fail to properly validate SSL and TLS certificates, and identifying applications using poor cryptography (such as block ciphers in ECB mode or CTR or CBC mode with a non-random IV). Such techniques would however not be able to identify flawed attempts at encryption which merely act as obfuscation, such as those shown in Section 5.3. Further work could perhaps look at the creation of safer cryptography libraries for developers wishing to encrypt data in another application, without having a clear understanding of the risks of naive implementation of encryption (such as lack of authenticated ciphertexts, order of encrypt and MAC operations, re-use of IVs, etc.). Current cryptographic libraries, even those which aim to be easy for developers to use, typically return keys to the calling application, and this keying material must be stored securely and properly in order for the encryption to be meaningful. A potential alternative approach would be for a cryptographic library to build itself around a secure key store implementation, such as that implemented in Chapter 7, such that applications calling the APIs would be returned merely a reference to the key, which need not be held securely, since the underlying key could only be obtained by the same application calling the cryptographic API. This would enforce strong keys, and prevent re-use of IVs, while also ensuring that developers have a simple API to use for encryption and decryption operations.

Bibliography

[1] B. Krebs. (2013, October) Adobe breach impacted at least 38 million users. [Online]. Available: http://krebsonsecurity.com/2013/10/adobe-breach-impacted-at-least-38-million-users/

[2] K. Thomas. (2010, December) Microsoft cloud data breach heralds things to come. [Online]. Available: http://www.pcworld.com/article/214775/microsoft_cloud_data_breach_sign_of_future.html

[3] L. Baker and J. Finkle. (2011, August) Sony PlayStation suffers massive data breach. [Online]. Available: http://www.reuters.com/article/us-sony-stoldendata-idUSTRE73P6WB20110426

[4] A. Tsotsis. (2014, December) Employee data breach the worst part of Sony hack. [Online]. Available: http://techcrunch.com/2014/12/16/hack-sony-twice-shame-on-sony/

[5] E. Harris and N. Perlroth. (2014, January) For Target, the breach numbers grow. [Online]. Available: http://www.nytimes.com/2014/01/11/business/target-breach-affected-70-million-customers.html

[6] B. McGuire. (2015, February) Insurer Anthem reveals hack of 80 million customer, employee accounts. ABC News. [Online]. Available: http://abcnews.go.com/Business/insurer-anthem-reveals-hack-80-million-customer-accounts/story?id=28737506

[7] A. Greenberg. OPM now admits 5.6m feds' fingerprints were stolen by hackers. [Online]. Available: http://www.wired.com/2015/09/opm-now-admits-5-6m-feds-fingerprints-stolen-hackers

[8] 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

[9] J. Finkle and K. Freifeld. (2014, April) Exclusive: U.S. states probing security breach at Experian unit. [Online]. Available: http://www.reuters.com/article/us-experian-databreach-idUSBREA321SL20140403

[10] T-Mobile. (2015, October) Frequently asked questions about the experian incident. [Online]. Available: https://www.t-mobile.com/landing/experian-data-breach-faq.html

[11] C. Baraniuk. (2015, August) Ashley Madison: Leaked accounts fallout deepens. [Online]. Available: http://www.bbc.co.uk/news/technology-34002915

[12] B. News. (2015, September) London clinic leaks HIV status of patients. [Online]. Available: http://www.bbc.co.uk/news/uk-england-london-34127740

[13] D. Bainbridge and G. Pearce, "The UK data protection act 1998 — data subjects' rights," *Computer Law & Security Review*, vol. 14, no. 6, pp. 401 – 406, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0267364998800575

[14] F. Li, A. Lai, and D. Ddl, "Evidence of advanced persistent threat: A case study of malware for political espionage," in *Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on.* IEEE, 2011, pp. 102–109.

[15] (2015, December) Agreement on Commission's EU data protection reform will boost digital single market. IP/15/6321. [Online]. Available: http://europa.eu/rapid/press-release_IP-15-6321_en.htm

[16] "Regulation (EU) 2016/679 of the European Parliament and of the Council," *Official Journal of the European Union*, May 2016, cELEX:32016R0679. [Online]. Available: http://data.europa.eu/eli/reg/2016/679/oj

[17] A. Tereshkin, "Evil maid goes after PGP whole disk encryption," in *Proceedings of the 3rd international conference on Security of information and networks.* ACM, 2010, pp. 2–2.

[18] AT&T. AT&T bug bounty program. [Online]. Available: https://bugbounty.att.com/home.php

[19] Github. Github security bug bounty. [Online]. Available: https://bounty.github.com/

[20] K. Zetter. (2015, October) Teen who hacked CIA director's email tells how he did it. [Online]. Available: http://www.wired.com/2015/10/hacker-who-broke-into-cia-director-john-brennan-email-tells-how-he-did-it/

[21] F. Babb. (2015, September) Amazon's aws dynamodb experiences outage, affecting netflix, reddit, medium, and more. [Online]. Available: http://venturebeat.com/2015/09/20/amazons-aws-outage-takes-down-netflix-reddit-medium-and-more/

[22] M. Honan, "How Apple and Amazon security flaws led to my epic hacking," *Wired Magazine*, 2012.

[23] T. Spring. (2016, March) Facebook password reset bug gave hackers access to any account. [Online]. Available: https://threatpost.com/facebook-password-reset-bug-gave-hackers-access-to-any-account/116651/

[24] (2013, April) UK adults taking online password security risks. Ofcom. [Online]. Available: http://media.ofcom.org.uk/news/2013/uk-adults-taking-online-password-security-risks/

[25] D. Irvine, J. Irvine, and S. K. Goo, "Sigmoid (x): Secure distributed network storage," in *Wireless World Research Forum 27*, 2011.

[26] Amazon. (2015, September) Summary of the Amazon DynamoDB service disruption and related impacts in the US-East region. [Online]. Available: https://aws.amazon.com/message/5467D2/

[27] Heroku. (2015, September) Elevated build and API error rates. [Online]. Available: https://status.heroku.com/incidents/811

[28] C. Babcock. (2012, July) Amazon outage hits Netflix, Heroku, Pinterest, Instagram. [Online]. Available: http://www.informationweek.com/cloud/infrastructure-as-a-service/amazon-outage-hits-netflix-heroku-pinterest-instagram/d/d-id/1105148

[29] (2014, January) Gmail went down and everyone panicked. [Online]. Available: http://www.huffingtonpost.com/2014/01/24/gmail-down_n_4660890.html

[30] D. Ma, "The business model of "software-as-a-service"," in *Services Computing, 2007. SCC 2007. IEEE International Conference on*, July 2007, pp. 701–702.

[31] Y. Rekhter, T. Li, and S. Hares, "A border gateway protocol 4 (BGP-4)," Internet Requests for Comments, RFC Editor, RFC 4271, January 2006. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4271.txt

[32] I. Clarke *et al.*, "A distributed decentralised information storage and retrieval system," Ph.D. dissertation, Master's thesis, University of Edinburgh, 1999.

[33] Merriam-Webster, ""centralized"," 2015. [Online]. Available: http://merriam-webster.com/dictionary/centralize

[34] Z. N. Peterson, M. Gondree, and R. Beverly, "A position paper on data sovereignty: the importance of geolocating data in the cloud," in *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, 2011.

[35] I. Agudo, D. Nuñez, G. Giammatteo, P. Rizomiliotis, and C. Lambrinoudakis, "Cryptography goes to the cloud," in *Secure and Trust Computing, Data Management, and Applications*. Springer, 2011, pp. 190–197.

[36] A. Bhimani, "Securing the commercial internet," *Commun. ACM*, vol. 39, no. 6, pp. 29–35, Jun. 1996. [Online]. Available: http://doi.acm.org/10.1145/228503.228509

[37] U. S. I. G. for Sysadmins. (2003, September) System administrators' code of ethics. [Online]. Available: https://www.usenix.org/lisa/system-administrators-code-ethics

[38] (2014, July) Glasgow becomes first scottish city to offer free wi-fi. [Online]. Available: https://www.glasgow.gov.uk/index.aspx?articleid=12292

[39] S. Choney. (2014, January) In Europe, there are 10 digital devices per household and the number will grow as smart computing takes hold. [Online]. Available: https://blogs.microsoft.com/firehose/2014/01/27/in-europe-there-are-10-digital-devices-per-household-and-the-number-will-grow-as-smart-computing-takes-hold/

[40] S. Chopra and L. White, "Privacy and artificial agents, or, is Google reading my email?" in *IJCAI*, 2007, pp. 1245–1250.

[41] J. P. Mello. (2013, June) Yahoo mail redesign becomes permanent, privacy issues surface. [Online]. Available: http://www.pcworld.com/article/2040642/yahoo-mail-redesign-becomes-permanent-privacy-issues-surface.html

[42] (2014, March) Microsoft admits reading Hotmail inbox of blogger. [Online]. Available: http://www.bbc.co.uk/news/business-26677607

[43] J. Pagliery. (2014, March) Microsoft defends its right to read your email. [Online]. Available: http://money.cnn.com/2014/03/21/technology/security/microsoft-email/index.html

[44] A. Hern. (2014, March) Yahoo, Google and Apple also claim right to read user emails. [Online]. Available: http://www.theguardian.com/technology/2014/mar/21/yahoo-google-and-apple-claim-right-to-read-user-emails

[45] L. Clark. (2014, February) Tim Berners-Lee: we need to re-decentralise the web. [Online]. Available: http://www.wired.co.uk/news/archive/2014-02/06/tim-berners-lee-reclaim-the-web

[46] J. Kopstein. (2013, December) The mission to decentralize the internet. [Online]. Available: http://www.newyorker.com/tech/elements/the-mission-to-decentralize-the-internet

[47] H. Shen, C.-Z. Xu, and G. Chen, “Cycloid: a constant-degree and lookup-efficient P2P overlay network,” *Performance Evaluation*, vol. 63, no. 3, pp. 195–216, 2006.

[48] V. Martins and E. Pacitti, “Dynamic and distributed reconciliation in p2p-dht networks,” *Euro-Par 2006 Parallel Processing*, pp. 337–349, 2006.

[49] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt, “P-grid: a self-organizing structured p2p system,” *ACM SIGMOD Record*, vol. 32, no. 3, pp. 29–33, 2003.

[50] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications,” *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.

[51] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, *A scalable content-addressable network.* ACM, 2001, vol. 31, no. 4.

[52] 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.

[53] L. Wang and J. Kangasharju, “Measuring large-scale distributed systems: case of bittorrent mainline DHT,” in *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on.* IEEE, 2013, pp. 1–10.

[54] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, “Freenet: A distributed anonymous information storage and retrieval system,” in *Designing Privacy Enhancing Technologies*. Springer, 2001, pp. 46–66.

[55] K. Bennett, T. Stef, C. Grothoff, T. Horozov, and I. Patrascu, “The GNet whitepaper,” 06/2002 2002.

[56] K. Bennett, C. Grothoff, T. Horozov, and I. Patrascu, “Efficient sharing of encrypted data,” in *Australasian Conference on Information Security and Privacy*. Springer, 2002, pp. 107–120.

[57] D. Kügler, “An analysis of GNUnet and the implications for anonymous, censorship-resistant networks,” in *International Workshop on Privacy Enhancing Technologies*. Springer, 2003, pp. 161–176.

[58] R. Anderson, *Security engineering*. John Wiley & Sons, 2008.

[59] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.

[60] D. Irvine, F. Hutchison, and S. Mücklisch, “Autonomous network,” pp. 1–9, 2010. [Online]. Available: http://docs.maidsafe.net/Whitepapers/pdf/AutonomousNetwork.pdf

[61] D. Irvine, “Self encrypting data,” pp. 1–4, 2010. [Online]. Available: http://docs.maidsafe.net/Whitepapers/pdf/SelfEncryptingData.pdf

[62] 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.

[63] A. Norberg and S. Siloti. (2015, September) Storing arbitrary data in the DHT. [Online]. Available: http://bittorrent.org/beps/bep_0044.html

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

[65] B. Bolosky, J. Douceur, D. Ely, and M. Theimer, “Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs,” in *Proceedings of the international conference on measurement and modeling of computer systems (SIGMETRICS)*.

Association for Computing Machinery, Inc., January 2000. [Online]. Available: https://www.microsoft.com/en-us/research/publication/feasibility-of-a-serverless-distributed-file-system-deployed-on-an-existing-set-of-desktop-pcs/

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

[67] D. Geer, "Reducing the storage burden via data deduplication," *Computer*, vol. 41, no. 12, pp. 15–17, 2008.

[68] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," *ACM Transactions on Storage (TOS)*, vol. 7, no. 4, p. 14, 2012.

[69] A. Mislove, "POST: a secure, resilient, cooperative messaging system," *Notes*, vol. 20, p. 22, 2003.

[70] F. Liu, Y. Sun, B. Li, and B. Li, "Quota: Rationing server resources in peer-assisted online hosting systems," in *Network Protocols, 2009. ICNP 2009. 17th IEEE International Conference on.* IEEE, 2009, pp. 103–112.

[71] 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.

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

[73] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry, "A fast file system for UNIX," *ACM Transactions on Computer Systems (TOCS)*, vol. 2, no. 3, pp. 181–197, 1984.

[74] P. Druschel and A. Rowstron, "Past: A large-scale, persistent peer-to-peer storage utility," in *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on.* IEEE, 2001, pp. 75–80.

[75] T.-W. J. Ngan, D. S. Wallach, and P. Druschel, *Peer-to-Peer Systems II: Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22, 2003. Revised Papers.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, ch. Enforcing Fair Sharing of Peer-to-Peer Resources, pp. 149–159. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-45172-3_14

[76] J. R. Douceur, "The sybil attack," in *Peer-to-peer Systems.* Springer, 2002, pp. 251–260.

[77] A. Juels and B. S. Kaliski Jr, “PORs: Proofs of retrievability for large files,” in *Proceedings of the 14th ACM conference on Computer and communications security*. Acm, 2007, pp. 584–597.

[78] Q. Zheng and S. Xu, “Secure and efficient proof of storage with deduplication,” *Proceedings of the second ACM conference on Data and Application Security and Privacy - CODASKY ’12*, p. 1, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2133601.2133603

[79] E. Guttman, “Service location protocol: Automatic discovery of IP network services,” *Internet Computing, IEEE*, vol. 3, no. 4, pp. 71–80, 1999.

[80] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, “An architecture for a secure service discovery service,” in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. ACM, 1999, pp. 24–35.

[81] C. Schmidt and M. Parashar, “A peer-to-peer approach to web service discovery,” *World Wide Web*, vol. 7, no. 2, pp. 211–229, 2004.

[82] R. Ranjan, L. Zhao, X. Wu, A. Liu, A. Quiroz, and M. Parashar, “Peer-to-peer cloud provisioning: Service discovery and load-balancing,” in *Cloud Computing*. Springer, 2010, pp. 195–217.

[83] H. Song, D. Cheng, A. Messer, and S. Kalasapur, “Web service discovery using general-purpose search engines,” in *Web Services, 2007. ICWS 2007. IEEE International Conference on*. IEEE, 2007, pp. 265–271.

[84] A. Dasgupta, A. Ghosh, R. Kumar, C. Olston, S. Pandey, and A. Tomkins, “The discoverability of the web,” in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 421–430.

[85] WhatsApp. How do I add contacts to WhatsApp? [Online]. Available: https://www.whatsapp.com/faq/android/21082107

[86] M. Marlinspike. (2014, January) The difficulty of private contact discovery. [Online]. Available: https://whispersystems.org/blog/contact-discovery/

[87] C. G. Addison, *A Treatise on the Law of Contracts*, 3rd ed. James Cockcroft & Company, 1875.

[88] Y. Cherdantseva and J. Hilton, "A reference model of information assurance & security," in *Availability, reliability and security (ares), 2013 eighth international conference on.* IEEE, 2013, pp. 546–555.

[89] B. Donohue. (2013, November) Cryptolocker is bad news. [Online]. Available: https://blog.kaspersky.com/cryptolocker-is-bad-news/3122/

[90] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, "Google Android: A comprehensive security assessment," *Security Privacy, IEEE*, vol. 8, no. 2, pp. 35–44, March 2010.

[91] Y. Y. Ng, H. Zhou, Z. Ji, H. Luo, and Y. Dong, "Which Android app store can be trusted in China?" in *Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual*, July 2014, pp. 509–518.

[92] K. Zetter. (2015, December) Secret code found in juniper's firewalls shows risk of government backdoors. [Online]. Available: http://www.wired.com/2015/12/juniper-networks-hidden-backdoors-show-the-risk-of-government-backdoors/

[93] R. B. Parker, "Definition of privacy, a," *Rutgers L. Rev.*, vol. 27, p. 275, 1973.

[94] R. Wilton, "Identity and privacy in the digital age," *International Journal of Intellectual Property Management*, vol. 2, no. 4, pp. 411–428, 2008.

[95] W. T. DeVries, "Protecting privacy in the digital age," *Berkeley Tech. LJ*, vol. 18, p. 283, 2003.

[96] *United States v. Jacobsen*, 1984, vol. 466.

[97] O. S. Kerr, "Fourth amendment seizures of computer data," *The Yale Law Journal*, pp. 700–724, 2010.

[98] W. J. Robison, "Free at what cost? cloud computing privacy under the stored communications act," *Georgetown Law Journal*, vol. 98, no. 4, 2010.

[99] D. J. Solove and P. M. Schwartz, "An overview of privacy law," 2015.

[100] OECD. (2013, July) 2013 OECD privacy guidelines. [Online]. Available: https://www.oecd.org/internet/ieconomy/privacy-guidelines.htm

[101] J. M. Fromholz, "European union data privacy directive, the," *Berk. Tech. LJ*, vol. 15, p. 461, 2000.

[102] P. Schwartz and J. R. Reidenberg, *Data privacy law: a study of United States data protection.* LEXIS law, 1996.

[103] P. P. Swire and R. E. Litan, *None of your business: world data flows, electronic commerce, and the European privacy directive.* Brookings Institution Press, 1998.

[104] W. J. Long and M. P. Quek, "Personal data privacy protection in an age of globalization: the US-EU safe harbor compromise," *Journal of European Public Policy*, vol. 9, no. 3, pp. 325–344, 2002.

[105] W. E. Agin, "The new regime for treatment of customer data in bankruptcy cases," *J. BANKR. LAW & PRAC.*, vol. 10, p. 365, 2001.

[106] "C‑362/14, Judgment of the Court (Grand Chamber) of 6 October 2015. Maximillian Schrems v Data Protection Commissioner," ECLI:EU:C:2015:650.

[107] C. L. Kunz, M. F. Del Duca, H. Thayer, and J. Debrow, "Click-through agreements: Strategies for avoiding disputes on validity of assent," *The Business Lawyer*, pp. 401–429, 2001.

[108] C. L. Kunz, J. E. Ottaviani, E. D. Ziff, J. M. Moringiello, K. M. Porter, and J. C. Debrow, "Browse-wrap agreements: Validity of implied assent in electronic form agreements," *The Business Lawyer*, pp. 279–312, 2003.

[109] R. H. Stern, "Shrink-wrap licenses of mass marketed software: Enforceable contracts or whistling in the dark," *Rutgers Computer & Tech. LJ*, vol. 11, p. 51, 1985.

[110] D. Shane. Facebook is "deliberately killing privacy", says Schneier. [Online]. Available: http://www.information-age.com/technology/security/1290603/facebook-is-%22deliberately-killing-privacy%22-says-schneier

[111] T. Louis. (2013, 31 August) How much is a user worth? [Online]. Available: http://www.forbes.com/sites/tristanlouis/2013/08/31/how-much-is-a-user-worth/

[112] Reuters, "Judge approves sale of RadioShack assets," May 2015.

[113] D. A. Munkittrick, "The legacy of the RadioShack bankruptcy and the importance of pii," October 2015. [Online]. Available: http://www.natlawreview.com/article/legacy-radioshack-bankruptcy-and-importance-pii

[114] M. Hiltzik. (2015, May) The RadioShack bankruptcy shows you can't trust a company's privacy pledge. [Online].

Available: http://www.latimes.com/business/hiltzik/la-fi-mh-radioshack-you-have-no-privacy-left-20150519-column.html

[115] B. Schaller. (2015, June) RadioShack bankruptcy case highlights value of consumer data. [Online]. Available: http://www.infolawgroup.com/2015/06/articles/privacy-law/radioshack-bankruptcy-case-highlights-value-of-consumer-data/

[116] E. London, D. Medine, and L. Mazzarella. FTC announces settlement with bankrupt website, toysmart.com, regarding alleged privacy policy violations. [Online]. Available: https://www.ftc.gov/news-events/press-releases/2000/07/ftc-announces-settlement-bankrupt-website-toysmartcom-regarding

[117] S. Ruwhof. Epic failure of Phone House & Dutch telecom providers to protect personal data: How I could access 12+ million records #phonehousegate. [Online]. Available: http://sijmen.ruwhof.net/weblog/608-personal-data

[118] T. Dierks and C. Allen, "The TLS protocol version 1.0," Internet Requests for Comments, RFC Editor, RFC 2246, January 1999. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2246.txt

[119] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, "TOTP: Time-based one-time password algorithm," Internet Requests for Comments, RFC Editor, RFC 6238, May 2011. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6238.txt

[120] C. Foxx. How to pick the perfect password. [Online]. Available: http://www.bbc.co.uk/news/technology-34221843

[121] U. Feige, A. Fiat, and A. Shamir, "Zero-knowledge proofs of identity," *Journal of cryptology*, vol. 1, no. 2, pp. 77–94, 1988.

[122] T. D. Wu, "The secure remote password protocol," in *NDSS*, vol. 98, 1998, pp. 97–111.

[123] C. Pedersen and D. Dahl, "Crypton: Zero-knowledge application framework."

[124] E. M. Tamil, A. H. Othman, S. A. Z. Abidin, M. Y. I. Idris, and O. Zakaria, "Password practices: A study on attitudes towards password usage among undergraduate students in klang valley, malaysia," *Journal of Advancement of Science & Arts*, vol. 3, pp. 37–42, 2007.

[125] D. Florencio and C. Herley, "A large-scale study of web password habits," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 657–666.

[126] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Consulted*, vol. 1, no. 2012, p. 28, 2008.

[127] A. Gervais, G. Karame, S. Capkun, and V. Capkun, "Is bitcoin a decentralized currency?" *IACR Cryptology ePrint Archive*, vol. 2013, p. 829, 2013.

[128] S. Srivastava and M. Novak. (2015, January) Bitcoin exchange Bitstamp suspends service after security breach. [Online]. Available: http://www.reuters.com/article/us-bitstamp-cybersecurity-idUSKBN0KF0UH20150106

[129] G. Cluley. (2015, February) Bitcoin exchange shuts down after suspected password breach. [Online]. Available: https://grahamcluley.com/2015/02/bitcoin-exchange-shuts-down/

[130] D. E. Sanger and J. H. Davis, "Data breach tied to China hits millions."

[131] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor, "Crying wolf: An empirical study of SSL warning effectiveness." in *USENIX Security Symposium*, 2009, pp. 399–416.

[132] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, "The most dangerous code in the world: validating ssl certificates in non-browser software," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 38–49.

[133] P. Schulz and D. Plohmann, "Android security-common attack vectors," *Rheinische Friedrich-Wilhelms-Universitat Bönn, Germany, Tech. Rep*, 2012.

[134] R. Gupta. (2014) Google mobile services. [Online]. Available: https://launchpad.motorolasolutions.com/APPFORUM2014/google_mobile_services.pdf

[135] A. Ludwig. (2014, April) Android security state of the union 2014. [Online]. Available: https://googleonlinesecurity.blogspot.co.uk/2015/04/android-security-state-of-union-2014.html

[136] L. Øverlier, "Data leakage from android smartphones," Ph.D. dissertation, Master's thesis, Norwegian Defence Research Establishment (FFI), 2012.

[137] A. Cortesi. (2014) mitmproxy. [Online]. Available: https://mitmproxy.org/

[138] S. Somogyi and A. Eijdenberg. (2015, September) Improved digital certificate security. [Online]. Available: https://googleonlinesecurity.blogspot.co.uk/2015/09/improved-digital-certificate-security.html

[139] A. Langley. (2015, March) Maintaining digital certificate security. [Online]. Available: https://googleonlinesecurity.blogspot.co.uk/2015/03/maintaining-digital-certificate-security.html

[140] ——. (2014, July) Maintaining digital certificate security. [Online]. Available: https://googleonlinesecurity.blogspot.co.uk/2014/07/maintaining-digital-certificate-security.html

[141] ——. (2013, December) Further improving digital certificate security. [Online]. Available: https://googleonlinesecurity.blogspot.co.uk/2013/12/further-improving-digital-certificate.html

[142] D. Fisher. (2011, September) Comodo, DigiNotar attacks expose crumbling foundation of ca system. [Online]. Available: https://threatpost.com/comodo-diginotar-attacks-expose-crumbling-foundation-ca-system-090211/75609/

[143] C. Soghoian and S. Stamm, "Certified lies: Detecting and defeating government interception attacks against SSL (short paper)," in *Financial Cryptography and Data Security*. Springer, 2012, pp. 250–259.

[144] C. Evans, C. Palmer, and R. Sleevi, "Public key pinning extension for HTTP," Internet Requests for Comments, RFC Editor, RFC 7469, April 2015. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7469.txt

[145] Google. (2013, October) Security enhancements in Android 4.4. [Online]. Available: https://source.android.com/security/enhancements/enhancements44.html

[146] N. Elenkov. (2012, December) Certificate pinning in Android 4.2. [Online]. Available: http://nelenkov.blogspot.co.uk/2012/12/certificate-pinning-in-android-42.html

[147] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *Security & Privacy, IEEE*, vol. 8, no. 6, pp. 40–47, 2010.

[148] Ç. K. Koç, *About Cryptographic Engineering*. Springer, 2009.

[149] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," *ACM Transactions on Storage (TOS)*, vol. 7, no. 4, p. 14, 2012.

[150] P.-L. Dubouilh, "Decentralised cloud project (bachelors report)," Tech. Rep., 2015.

[151] D. J. Bernstein, "The salsa20 family of stream ciphers," in *New stream cipher designs*. Springer, 2008, pp. 84–97.

[152] D. J. Bernstein, T. Lange, and P. Schwabe, "The security impact of a new cryptographic library," in *Progress in Cryptology–LATINCRYPT 2012*. Springer, 2012, pp. 159–176.

[153] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman, "Mining your Ps and Qs: Detection of widespread weak keys in network devices." in *USENIX Security Symposium*, 2012, pp. 205–220.

[154] D. Irvine, "Maidsafe distributed hash table," pp. 1–5, 2010. [Online]. Available: http://docs.maidsafe.net/Whitepapers/pdf/MaidSafeDistributedHashTable.pdf

[155] E. Sit and R. Morris, "Security considerations for peer-to-peer distributed hash tables," in *Peer-to-Peer Systems*. Springer, 2002, pp. 261–269.

[156] F. Jacob, J. Mittag, and H. Hartenstein, "A security analysis of the emerging p2p-based personal cloud platform maidsafe," in *Trustcom/BigDataSE/ISPA, 2015 IEEE*, vol. 1. IEEE, 2015, pp. 1403–1410.

[157] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, "Inferring internet denial-of-service activity," *ACM Transactions on Computer Systems (TOCS)*, vol. 24, no. 2, pp. 115–139, 2006.

[158] T. Duong and J. Rizzo, "Flickr's API signature forgery vulnerability," 2009.

[159] D. J. Bernstein, "Salsa20 design," 2005.

[160] J.-P. Aumasson, S. Fischer, S. Khazaei, W. Meier, and C. Rechberger, "New features of latin dances: analysis of salsa, chacha, and rumba," in *Fast Software Encryption*. Springer, 2008, pp. 470–488.

[161] N. Mouha and B. Preneel, "Towards finding optimal differential characteristics for arx: Application to salsa20," Cryptology ePrint Archive, Report 2013/328, Tech. Rep., 2013.

[162] D.J. Bernstein, “Which estream ciphers have been broken?” *Ecrypt report*, vol. 10, p. 2008, 2008.

[163] Z. Wilcox-O’Hearn. (2012, March) zfec. [Online]. Available: https://pypi.python.org/pypi/zfec

[164] D. Irvine, ““peer to peer” public key infrastructure,” pp. 1–6, 2011.

[165] M. Jones, J. Bradley, and N. Sakimura, “Json web token (JWT),” Internet Requests for Comments, RFC Editor, RFC 7519, May 2015. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7519.txt

[166] (2015, April) Cve-2015-2951. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-2951

[167] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, “High-speed high-security signatures,” *Journal of Cryptographic Engineering*, pp. 1–13, 2012.

[168] N. Koblitz and A. Menezes, “Another look at security definitions.” *IACR Cryptology ePrint Archive*, vol. 2011, p. 343, 2011.

[169] W. Lin, J. Shi, H. Tian, and Q. Cui, “Robust and scalable mobility support for real-time applications,” in *Wireless Communications and Networking Conference, 2008. WCNC 2008. IEEE.* IEEE, 2008, pp. 3219–3224.

[170] S. Baset, H. Schulzrinne, E. Shim, and K. Dhara, “Requirements for SIP-based peer-to-peer internet telephony,” *IETF draft-baset-sipping-p2preq-00*, 2005.

[171] S. Buchegger, D. Schiöberg, L.-H. Vu, and A. Datta, “Peerson: P2P social networking: early experiences and insights,” in *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems.* ACM, 2009, pp. 46–52.

[172] V. Bruno, A. Tam, and J. Thom, “Characteristics of web applications that affect usability: a review,” in *Proceedings of the 17th Australia conference on Computer-Human Interaction: Citizens Online: Considerations for Today and the Future.* Computer-Human Interaction Special Interest Group (CHISIG) of Australia, 2005, pp. 1–4.

[173] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris, “Designing a dht for low latency and high throughput.” in *NSDI*, vol. 4, 2004, pp. 85–98.

[174] L. Harn and H.-Y. Lin, “Key management for decentralized computer network services,” *Communications, IEEE Transactions on*, vol. 41, no. 12, pp. 1777–1779, 1993.

[175] S. A. Baset and H. Schulzrinne, “An analysis of the Skype peer-to-peer internet telephony protocol,” *arXiv preprint cs/0412017*, 2004.

[176] G. Kreitz and F. Niemelä, “Spotify–large scale, low latency, P2P music-on-demand streaming.” in *Peer-to-Peer Computing*, 2010, pp. 1–10.

[177] M. Goldmann and G. Kreitz, “Measurements on the spotify peer-assisted music-on-demand streaming system,” in *Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on.* IEEE, 2011, pp. 206–211.

[178] D. Goodin. (2012, May) Skype replaces P2P supernodes with Linux boxes hosted by Microsoft. [Online]. Available: http://arstechnica.com/business/2012/05/skype-replaces-p2p-supernodes-with-linux-boxes-hosted-by-microsoft/

[179] R. Dillet. (2014, April) Spotify removes peer-to-peer technology from its desktop client. Techcrunch. Retrieved 5th March 2015. [Online]. Available: http://techcrunch.com/2014/04/17/spotify-removes-peer-to-peer-technology-from-its-desktop-client/

[180] G. Greenwald, E. MacAskill, L. Poitras, S. Ackerman, and D. Rushe. (2013, July) Microsoft handed the NSA access to encrypted messages. [Online]. Available: http://www.theguardian.com/world/2013/jul/11/microsoft-nsa-collaboration-user-data

[181] L. Vaas. (2013, May) Microsoft is reading Skype messages. [Online]. Available: https://nakedsecurity.sophos.com/2013/05/22/microsofts-reading-skype-messages/

[182] M. Kaufman. (2013, June) From the principal architect – Skype / NSA (mailing list post). [Online]. Available: https://www.listbox.com/member/archive/247/2013/06/sort/time_rev/page/1/entry/6:271/20130623090855:0B714E0A-DC06-11E2-9F35-8CD4CCA160A2/

[183] J. O’Toole. (2014, February) Mobile apps overtake PC internet usage in U.S. CNN. [Online]. Available: http://money.cnn.com/2014/02/28/technology/mobile/mobile-apps-internet/index.html

[184] (2014, September) The state of broadband 2014: Broadband for all. The Broadband Commission. [Online]. Available: http://www.broadbandcommission.org/Documents/reports/bb-annualreport2014.pdf

[185] H. Pucha, S. M. Das, and Y. C. Hu, "How to implement DHTs in mobile ad hoc networks," in *Proc. of the 10th ACM International Conference on Mobile Computing and Network (MobiCom 2004)*, 2004.

[186] I. Kelenyi and B. Forstner, "Distributed hash table on mobile phones," in *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE.* IEEE, 2008, pp. 1226–1227.

[187] I. Kelényi and J. K. Nurminen, "Energy aspects of peer cooperation measurements with a mobile DHT system," in *Communications Workshops, 2008. ICC Workshops' 08. IEEE International Conference on.* IEEE, 2008, pp. 164–168.

[188] M. Bauer, M. Coatsworth, and J. Moeller, "NANSA: A no-attribution, no-sleep battery exhaustion attack for portable computing devices."

[189] H. Zhang, A. Goel, and R. Govindan, "Incrementally improving lookup latency in distributed hash table systems," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, pp. 114–125, Jun. 2003. [Online]. Available: http://doi.acm.org/10.1145/885651.781042

[190] J. Falkner, M. Piatek, J. P. John, A. Krishnamurthy, and T. Anderson, "Profiling a million user DHT," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement.* ACM, 2007, pp. 129–134.

[191] S. Guha and P. Francis, "Characterization and measurement of TCP traversal through NATs and firewalls," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement.* USENIX Association, 2005, pp. 18–18.

[192] C. I. Jaramillo, G. Richard, S. Sperry, and W. Patterson, "An implementation of a zero-knowledge protocol for a secure network login procedure," in *Southeastcon'89. Proceedings. Energy and Information Technologies in the Southeast., IEEE.* IEEE, 1989, pp. 197–201.

[193] (2012, April) Android issue tracker - different passwords for encryption and screen lock. Google Inc. [Online]. Available: https://code.google.com/p/android/issues/detail?id=29468

[194] Google Inc. (2016, August) Android 7.0 behavior changes. [Online]. Available: https://developer.android.com/about/versions/nougat/android-7.0-changes.html

[195] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, “Checking app behavior against app descriptions,” in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 1025–1035.

[196] A. Mahajan, M. Dahiya, and H. Sanghvi, “Forensic analysis of instant messenger applications on android devices,” *arXiv preprint arXiv:1304.4915*, 2013.

[197] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, “Why Eve and Mallory love Android: An analysis of Android SSL (in)security,” in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS ’12. New York, NY, USA: ACM, 2012, pp. 50–61. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382205

[198] simple app 2016. (2016, January) Vlocker-hide videos. [Online]. Available: https://play.google.com/store/apps/details?id=com.simpleapp.vlocker

[199] S. Ltd. (2015, October) Hide pictures & videos - fotox. [Online]. Available: https://play.google.com/store/apps/details?id=com.smsrobot.photox

[200] H. Apps. (2015, December) Video locker - hide videos. [Online]. Available: https://play.google.com/store/apps/details?id=com.handyapps.videolocker

[201] Handy Apps. (2015, December) Hide photos in photo locker. [Online]. Available: https://play.google.com/store/apps/details?id=com.handyapps.photoLocker

[202] ——. (2015, December) Password locker - encrypt data. [Online]. Available: https://play.google.com/store/apps/details?id=com.handyapps.passwordlocker

[203] NewSoftwares.net. (2016, January) Video locker - hide videos. [Online]. Available: https://play.google.com/store/apps/details?id=net.newsoftwares.videolockeradvanced

[204] T. Mobile. (2015, December) Gallery vault-hide video & photo. [Online]. Available: https://play.google.com/store/apps/details?id=com.thinkyeah.galleryvault

[205] MobilDev. (2014, November) Encrypt file free. [Online]. Available: https://play.google.com/store/apps/details?id=com.acr.encryptfilefree

[206] A. Kerckhoffs, “La cryptographie militaire (military cryptography),⊠ j,” *Sciences Militaires (J. Military Science, in French)*, 1883.

[207] FTC. (2013, February) Mobile app developers: Start with security. [Online]. Available: https://www.ftc.gov/tips-advice/business-center/guidance/mobile-app-developers-start-security

[208] C. Donnelly. (2012, September) Android app maker fined £50,000 for misleading consumers. [Online]. Available: http://www.itpro.co.uk/642616/android-app-maker-fined-50000-for-misleading-consumers

[209] Department for Busienss, Innovation & Skills. (2015, October) New rights for consumers when buying digital content. [Online]. Available: https://www.gov.uk/government/news/new-rights-for-consumers-when-buying-digital-content

[210] M.-F. Chang and C.-H. Chen, "Temporary call-back telephone number service," in *Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on*, March 2012, pp. 314–318.

[211] Snapchat. (2014, January) Find friends abuse. [Online]. Available: http://blog.snapchat.com/post/72013106599/find-friends-abuse

[212] E. De Cristofaro, M. Manulis, and B. Poettering, "Private discovery of common social contacts," *International Journal of Information Security*, vol. 12, no. 1, pp. 49–65, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10207-012-0183-4

[213] M. Nagy, E. De Cristofaro, A. Dmitrienko, N. Asokan, and A.-R. Sadeghi, "Do I know you?: Efficient and privacy-preserving common friend-finder protocols and applications," in *Proceedings of the 29th Annual Computer Security Applications Conference*, ser. ACSAC '13. New York, NY, USA: ACM, 2013, pp. 159–168. [Online]. Available: http://doi.acm.org/10.1145/2523649.2523668

[214] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 1, pp. 379–423, 623–656, 1948.

[215] WhatsApp. (2012, July) Whatsapp legal info. [Online]. Available: https://www.whatsapp.com/legal/#Privacy

[216] C. Percival, "The scrypt password-based key derivation function," 2012. [Online]. Available: https://www.tarsnap.com/scrypt/scrypt.pdf

[217] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Advances in Cryptology—CRYPTO'96*. Springer, 1996, pp. 1–15.

[218] C. Percival and S. Josefsson. (2015, January) The scrypt password-based key derivation function. IETF. [Online]. Available: http://tools.ietf.org/html/draft-josefsson-scrypt-kdf-02

[219] H. Orman, “Twelve random characters: Passwords in the era of massive parallelism,” *Internet Computing, IEEE*, vol. 17, no. 5, pp. 91–94, Sept 2013.

[220] R. Entner, “International comparisons: The handset replacement cycle,” Recoon Analytics, Tech. Rep., June 2011.

[221] R. Dunbar, “How many ”friends” can you really have?” *Spectrum, IEEE*, vol. 48, no. 6, pp. 81–83, 2011.

[222] R. Farahbakhsh, X. Han, A. Cuevas, and N. Crespi, “Analysis of publicly disclosed information in facebook profiles,” in *Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on*, Aug 2013, pp. 699–705.

[223] C. Serrano, B. Garriga, J. Velasco, J. Urbano, S. Tenorio, and M. Sierra, “Latency in broad-band mobile networks,” in *Vehicular Technology Conference, 2009. VTC Spring 2009. IEEE 69th*, April 2009, pp. 1–7.

[224] Identity, *Oxford English Dictionary*. Oxford University Press, 2015.

[225] L. Chun-Li, S. Hung-Min, and T. Hwang, “Attacks and solutions on strong-password authentication,” *IEICE transactions on communications*, vol. 84, no. 9, pp. 2622–2627, 2001.

[226] D. Boneh and M. Franklin, “Identity-based encryption from the weil pairing.” Springer-Verlag, 2001, pp. 213–229.

[227] Samsung unveils Galaxy S5 to focus on what matters most to consumers. [Online]. Available: http://www.samsungmobilepress.com/2014/02/25/Samsung-unveils-Galaxy-S5-to-focus-on-what-matters-most-to-consumers-1

[228] Sony unveils next-generation smartphone camera with Xperia Z5 and Xperia Z5 Compact. [Online]. Available: http://blogs.sonymobile.com/press_release/sony-unveils-xperia-z5-series/

[229] OnePlus 2 technology. [Online]. Available: https://oneplus.net/uk/2/technology

[230] P. Corcoran, "Biometrics and consumer electronics: A brave new world or the road to dystopia? [soapbox]," *Consumer Electronics Magazine, IEEE*, vol. 2, no. 2, pp. 22–33, April 2013.

[231] Motorola. Do I need to use my four to six digit code to secure my phone? [Online]. Available: https://motorola-global-portal.custhelp.com/app/answers/detail/a_id/62441/~/atrix---fingerprint-smart-sensor

[232] N. L. Clarke and S. M. Furnell, "Authentication of users on mobile telephones–a survey of attitudes and practices," *Computers & Security*, vol. 24, no. 7, pp. 519–527, 2005.

[233] S. M. Furnell, P. Dowland, H. Illingworth, and P. L. Reynolds, "Authentication and supervision: A survey of user attitudes," *Computers & Security*, vol. 19, no. 6, pp. 529–539, 2000.

[234] L. A. Jones, A. I. Antón, and J. B. Earp, "Towards understanding user perceptions of authentication technologies," in *Proceedings of the 2007 ACM workshop on Privacy in electronic society*. ACM, 2007, pp. 91–98.

[235] M. Harbach, E. von Zezschwitz, A. Fichtner, A. De Luca, and M. Smith, "It's a hard lock life: A field study of smartphone (un) locking behavior and risk perception," in *Symposium on Usable Privacy and Security (SOUPS)*, 2014.

[236] (2013, September) Chaos computer club breaks Apple TouchID. [Online]. Available: http://www.ccc.de/en/updates/2013/ccc-breaks-apple-touchid

[237] A. Hern. (2014, December) Hacker fakes German minister's fingerprints using photos of her hands. [Online]. Available: http://www.theguardian.com/technology/2014/dec/30/hacker-fakes-german-ministers-fingerprints-using-photos-of-her-hands

[238] D. Goodin. (2008, March) Get your German interior minister's fingerprint here. [Online]. Available: http://www.theregister.co.uk/2008/03/30/german_interior_minister_fingerprint_appropriated/

[239] Being arrested: your rights. [Online]. Available: https://www.gov.uk/arrested-your-rights/giving-fingerprints-photographs-and-samples

[240] Office of biometric identity management. [Online]. Available: http://www.dhs.gov/obim

[241] (2014, October) Virginia judge: Police can demand a suspect unlock a phone with a fingerprint. [Online]. Available: http://arstechnica.com/tech-policy/2014/10/virginia-judge-police-can-demand-a-suspect-unlock-a-phone-with-a-fingerprint/

[242] "Fingerprints on mobile devices: Abusing and leaking," in *Black Hat conference*, 2015. [Online]. Available: https://www.blackhat.com/docs/us-15/materials/us-15-Zhang-Fingerprints-On-Mobile-Devices-Abusing-And-Leaking-wp.pdf

[243] (2015, August) Full TrustZone exploit for MSM8974. [Online]. Available: http://bits-please.blogspot.co.uk/2015/08/full-trustzone-exploit-for-msm8974.html

[244] D. Rosenberg, "QSEE TrustZone kernel integer over flow vulnerability," in *Black Hat conference*, 2014.

[245] D. Shen, "Exploiting Trustzone on Android," in *Black Hat conference*, 2015.

[246] (2010, May) Poland installs Europe's first biometric fingerprint-scanning ATM machine. [Online]. Available: http://www.popsci.com/technology/article/2010-05/poland-installs-europes-first-biometric-fingerprint-scanning-atm-machines

[247] I. Ion and B. Dragovic, "Don't trust POS terminals! Verify in-shop payments with your phone," *Proceedings of SMPU*, vol. 8, 2010.

[248] Bank of Melbourne enables fingerprint login for Apple iOS users. [Online]. Available: http://www.arnnet.com.au/article/555819/bank-melbourne-enables-fingerprint-login-apple-ios-users/

[249] M. Yousefpor, J.-M. Bussat, B. B. Lyon, G. Gozzini, S. P. Hotelling, and D. Setlak, "Fingerprint sensor in an electronic device," August 2014, US Patent App. 14/451,076.

[250] Apple. Apple pay. [Online]. Available: https://www.apple.com/apple-pay/

[251] *Protection of Freedoms Act 2012*. [Online]. Available: http://www.legislation.gov.uk/ukpga/2012/9

[252] M. El-Abed, R. Giot, B. Hemery, and C. Rosenberger, "A study of users' acceptance and satisfaction of biometric systems," in *Security Technology (ICCST), 2010 IEEE International Carnahan Conference on*, Oct 2010, pp. 170–178.

[253] J. Clulow, *On the Security of PKCS #11.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 411–425. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-45238-6_32

[254] J. Hertl, "Verifying and improving cryptographic key security in PKCS# 11 implementations," Ph.D. dissertation, Masarykova univerzita, Fakulta informatiky, 2014.

[255] A. Aendenroomer and S. Huang, "Dynamic flash-memory allocation for smart-cards: how to cope with limited space (in a short life)," in *Industrial Informatics, 2007 5th IEEE International Conference on*, vol. 2, June 2007, pp. 835–840.

[256] W. Koch. The OpenPGP card. [Online]. Available: http://www.g10code.com/p-card.html

[257] D. Boger, L. Barreto, J. Fraga, P. Urien, H. Aissaoui, A. Santos, and G. Pujolle, "User-centric identity management based on secure elements," in *Computers and Communication (ISCC), 2014 IEEE Symposium on*, June 2014, pp. 1–6.

[258] M. Bellare, J. Kilian, and P. Rogaway, "The security of the cipher block chaining message authentication code," *Journal of Computer and System Sciences*, vol. 61, no. 3, pp. 362–399, 2000.

[259] J. Daemen and V. Rijmen, "The block cipher Rijndael," in *Smart Card Research and Applications.* Springer, 2000, pp. 277–284.

[260] P. Chown, "Advanced encryption standard (aes) ciphersuites for transport layer security (tls)," Internet Requests for Comments, RFC Editor, RFC 3268, June 2002. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3268.txt

[261] ISO, "Identification cards—Integrated circuit cards—Part 4: Organization, security and commands for interchange," International Organization for Standardization, Geneva, Switzerland, ISO 7816—4:2005, 2005.

[262] R. Anderson and R. Needham, "Robustness principles for public key protocols," in *Advances in Cryptology—CRYPT0'95.* Springer, 1995, pp. 236–247.

[263] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Examining smart-card security under the threat of power analysis attacks," *IEEE transactions on computers*, vol. 51, no. 5, pp. 541–552, 2002.

[264] V. Cortier, S. Delaune, and G. Steel, "A formal theory of key conjuring," in *20th IEEE Computer Security Foundations Symposium (CSF'07).* IEEE, 2007, pp. 79–96.

[265] M. Bond, “Attacks on cryptoprocessor transaction sets,” in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2001, pp. 220–234.

[266] D. M’Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen, “HOTP: An hmac-based one-time password algorithm,” Internet Requests for Comments, RFC Editor, RFC 4226, December 2005.

[267] Google Inc. (2016, September) Google authenticator. [Online]. Available: https://play.google.com/store/apps/details?id=com.google.android.apps.authenticator2

[268] Yubico. (2013, September) Android application for TOTP with yubikey neo. [Online]. Available: https://github.com/Yubico/yubitotp-android/

[269] H. Krawczyk, M. Bellare, and R. Canetti, “HMAC: Keyed-hashing for message authentication,” Internet Requests for Comments, RFC Editor, RFC 2104, February 1997. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2104.txt

[270] J. Aas. (2015, October) Let’s encrypt is trusted. [Online]. Available: https://letsencrypt.org/2015/10/19/lets-encrypt-is-trusted.html

[271] R. Barnes, J. Hoffman-Andrews, and J. Kasten. (2016, July) Automatic certificate management environment (ACME) draft 3. IETF. [Online]. Available: https://tools.ietf.org/html/draft-ietf-acme-acme-03

[272] Let’s Encrypt. (2016, September) How it works. [Online]. Available: https://letsencrypt.org/how-it-works/

[273] (2015, September) What happens when an account key is stolen? [Online]. Available: https://community.letsencrypt.org/t/what-happens-when-an-account-key-is-stolen/861

[274] J. Jonsson and B. Kaliski, “Public-key cryptography standards (PKCS) #1: Rsa cryptography specifications version 2.1,” Internet Requests for Comments, RFC Editor, RFC 3447, February 2003. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3447.txt

[275] P. Svenda. Javacard algorithms support test. [Online]. Available: http://www.fi.muni.cz/~xsvenda/jcsupport.html

[276] D. Roesler. acme-tiny. [Online]. Available: https://github.com/diafygi/acme-tiny

[277] N. Itoi and P. Honeyman, “Practical security systems with smartcards,” in *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, 1999, pp. 185–190.

[278] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer Science & Business Media, 2008, vol. 31.

[279] P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” in *Advances in Cryptology — CRYPTO' 99*, ser. Lecture Notes in Computer Science, M. Wiener, Ed. Springer Berlin Heidelberg, 1999, vol. 1666, pp. 388–397. [Online]. Available: http://dx.doi.org/10.1007/3-540-48405-1_25

[280] A. Lemarechal, “Introduction to fault attacks on smartcard,” in *11th IEEE International On-Line Testing Symposium*, July 2005, pp. 116–.

[281] E. Poll. (2015) Fault injections: Defensive coding for smartcards & fault attacks on mainstream hardware. [Online]. Available: https://www.cs.ru.nl/E.Poll/hw/slides/8_Faults_DefensiveCoding.pdf

[282] V. Lorenc, T. Smolka, and P. Švenda, “Automatic source code transformations for strengthening practical security of smart card applications,” 2010.