

Table of Contents

1	Introduction	3
1.1	Motivation	3
1.2	Project Aims	4
1.3	Overview of Report	4
2	Background Material	6
2.1	Android Security Features	6
2.2	Dropbox API	7
2.3	Review of Related Work	8
2.4	Review of Other Products	8
3	Analysis and Design	10
3.1	Overview	10
3.2	Problem Analysis	11
3.3	Document Storage	12
3.4	Cryptographic Design	13
3.5	Android Prototype	15
4	Implementation	17
4.1	Development Environment	17
4.2	Initial Development	18
4.3	SecurelyShare Prototype	18
4.4	Implementation Challenges	19
4.5	Testing	19
5	Project Management	20
6	Evaluation	21
6.1	Security Evaluation	21
6.2	Product Evaluation	21

6.3	Process Evaluation	21
6.4	Future Work	21
7	Conclusion	22
8	Implementation	23
8.1	Development Environment	23
8.2	Initial Development	24
8.3	Testing	25
9	Project Management	27

Chapter 1

Introduction

1.1 Motivation

As the popularity of mobile communications devices increases, there is a growing tendency to use these as a convenient means of reviewing and revising documents on-the-move. Where these documents are of a confidential nature, particular attention must be paid to the fact that mobile devices are more vulnerable to compromise than traditional desktops, which are usually more extensively protected by the security measures implemented as part of an organization's internal network.

There are multiple mechanisms for keeping files secure on company servers whilst allowing employees the necessary permissions to work collaboratively with sensitive data as required. As the mobile device culture becomes more prevalent in the workplace, the addition of Mobile Device Management (MDM) applications empowers users to also access corporate data via their mobile devices whilst still allowing IT departments to retain a degree of control over data security.

Thus, it is acknowledged that maintaining the security of confidential documents can be challenging, even with the weight of a corporate IT infrastructure behind it. In this project, we seek to address the issue of allowing groups of users from different organizations (i.e. with no shared IT infrastructure) to collaborate securely on confidential documents and furthermore, to access these documents via a smartphone or tablet computer whilst minimizing the risk of exposing sensitive information to a potential attacker.

1.2 Project Aims

The primary aim of this project was to implement a scheme to facilitate secure sharing of confidential documents between a number of collaborators (typically less than 15), subject to the following constraints:-

- Groups are self-organizing and represent multiple organizations, hence they cannot draw on the support of any central IT services.
- The documents involved are confidential in nature and hence should be encrypted both in transit and at rest.
- Group members wish to be able to access documents on a mobile device running the Android operating system (which may involve use of public wifi) without compromising security
- The solution devised should use only well-tested cryptographic techniques and standard libraries and should minimize the amount of trust to be placed in a third-party.

Our secondary objective was to gain an understanding of the basic features of the Android platform, explore some of the techniques involved in developing for mobile devices and some of the challenges encountered in working in this type of event-driven environment.

In pursuit of these aims we developed a solution called SecurelyShare, consisting of a detailed design of the security components of the system and a prototype Android application to provide a platform on which to implement and evaluate the various security features. It was acknowledged that, in a live setting, documents would usually originate on a PC rather than on a tablet device and thus the system would also need to have a PC-based component. However, within the time constraints of the project it was considered infeasible to develop a fully featured system; our solution is submitted rather as a 'proof of concept'.

1.3 Overview of Report

The subsequent chapters of this report will deal with the design, implementation and evaluation of the project. Chapter 2 introduces some of the background material on key technologies used and presents an overview of the Android applications reviewed as part of our preliminary research. In the light of this research, Chapter 3 presents a detailed analysis of the problem, defines the threat model against which we are attempting to defend, and provides an outline of the final solution design. Chapters 8 and 9 provide details of the implementation, testing and management of the project, Chapter ?? reviews the security

features and finally Chapter ?? presents and evaluation of overall success of project and gives recommendations for future work.

Chapter 2

Background Material

In this chapter we will introduce some key aspects of the android architecture and its security features. We will also examine the features offered by the Dropbox API and finally we will look briefly at some of the commercial applications which were reviewed as part of our initial research and which offer some features similar to Securely Share.

2.1 Android Security Features

Android provides an open source platform and application environment for mobile devices. It has a layer based architecture whose foundational component is the Android Operating System. Based on the tried and tested Linux 2.6 kernel and modified by Google to include some additional features, it is the comprehensive user permissions model inherent in Linux that is responsible for providing the separation between applications that is a key feature of Android security. At installation time, each application is assigned a unique user ID (UID); at runtime each application is run as a separate process and as a separate user with its given UID. This creates an Application Sandbox, protecting any resources belonging to that application (memory space, files, etc.) from being accessed by another application unless specifically permitted to do so by the developer.

May need to expand this a bit further but not quite sure how much detail is required and what is extraneous padding

2.2 Dropbox API

Dropbox is a cloud storage service that also offers users automatic backup facilities, file synchronization across devices, and the ability to share files with other users. It provides multi-platform client applications plus a series of public APIs that enable different subsets of the Dropbox functionality to be integrated into third-party applications.

On the Android platform, Dropbox offers three APIs, described on its website as follows:-

Core The Core API includes the most comprehensive functionality including features such as search, file restore, etc. Although more complex than either of the other two to implement, it is often more suitable when developing server-based apps.

Datastore The Datastore API provides a means of storing and synchronizing structured data like contacts, to-do items, and game states across all the user's devices.

Sync The Sync API provides a file system for accessing and writing files to the user's Dropbox. The Sync API manages the process of synchronizing file changes to Dropbox and can also provide the app with notification when changes are made to files stored on the server.

For the purposes of the SecurelyShare app, although it offers the most basic interface to the Dropbox server, the Sync API was deemed to support both the functionality required for the prototype and some additional facilities which could be implemented at a later date in order to improve the overall user experience.

Each app on the Dropbox Platform needs to be registered in the App Console and the developer needs to select which permissions the app requires. These permissions determine the type of data that the app can access in the user's Dropbox. For the Sync API, three levels of permission were available:

1. App folder: this creates a folder in the user's Dropbox with the same name as the app, all files relating to the app are kept here and access is restricted to this folder and its subfolders.
2. File type: the app is given access to the user's entire Dropbox but is restricted only to seeing files of certain types (documents, images, ebooks, etc.)
3. Full Dropbox: the app is given unrestricted access to the user's Dropbox

Developers are also able to submit a request to use a custom file extension with the File type permission.

2.3 Review of Related Work

was intending to include a summary of all the material relating to key distribution that I reviewed - not sure if I am going to be able to do this now

2.4 Review of Other Products

One of the security claims of cloud storage provider, Dropbox, is that user data stored on their servers is fragmented and encrypted using 256-bit AES. However, although users may feel reassured by these claims, it is also to be noted that since this encryption is applied server-side, the servers also have access to the keys required to decrypt this information. Furthermore, the Dropbox privacy policy states that,

"We may disclose your information to third parties if we determine that such disclosure is reasonably necessary to (a) comply with the law; (b) protect any person from death or serious bodily injury; (c) prevent fraud or abuse of Dropbox or our users; or (d) protect Dropbox's property rights."

It may therefore come as little surprise that there are an increasing number of client-side encryption applications available that integrate with Dropbox and other similar cloud services to ensure that, even if files are disclosed, the service providers would have no access to decryption keys. Although there are a significant number of encryption applications available for Android, most were discarded for consideration here as they are primarily designed for a single user to encrypt files prior to cloud storage and had little or no support for key sharing.

When trying to evaluate these products there appeared to be a marked absence of robust reviews from respected members of the security fraternity; many of the review that were available seemed to focus more on the usability of applications rather than on their security features. We chose three commercially produced applications to consider in greater depth, Boxcryptor, Viivo and SafeMonk. These were selected partly because they seemed to attract most mentions (although that may simply be due to the fact that they were commercial products and hence had larger marketing budgets), but also because we were able to find some level of technical description of the security protocols used. There was a common trend among the examined applications to use AES for encryption of user files and some implementation of RSA to manage the key exchange, All three applications used a central

server for key management, although each was keen to stress that they had no access to key material that would enable them to decrypt user files.

need to put some more discussion of how they manage key sharing

There justification for storing the user's private key on the server was that it would allow the user to install the application on multiple devices and they both used password-based encryption (PBE) in order to secure the key on the server. Although the servers use key-stretching algorithms to derive the encryption keys, we should always be aware that users can be very poor at choosing strong passwords and at keeping them secure. (In a study of password reuse, Bonneau Das et al. (2014) observed that 43 percent of users reused a password across multiple sites)

Chapter 3

Analysis and Design

In this chapter we will present some of the salient points from our initial analysis of the problem, detail the threat model against which we are trying to protect and any significant assumptions made which influenced our design decisions, and finally outline our solution design. It should be noted at this point that we had absolutely no prior experience of developing for (or even using) an Android device, therefore this phase was highly iterative. As our familiarity and understanding of the capabilities of Android platform increased and as the design evolved, we were able to revisit these assumptions and the threat model in order to strengthen the security of the overall system. It is for this reason that we feel that presenting this material in a single chapter is more reflective of the underlying process.

3.1 Overview

At the outset, the following key areas were identified that would need further research:

- finalisation of the threat model and main assumptions
- mechanism to be used for storage and distribution of shared documents
- security design, including any key generation and distribution schemes
- processes involved in setting up or managing groups
- design of the prototype application, including any steps required to protect sensitive key material

3.2 Problem Analysis

Assumptions

As outlined in the introduction, Securely Share is intended for use by small, autonomous groups (typically of the order of 3-15 members) with a largely static membership. This factor was significant in enabling us to discard a number of complex key distribution protocols which were designed for much larger or more volatile groups. It was also envisaged that groups would have lifetime measured in months, rather than days, hence a small administrative overhead in setting up the group could be considered acceptable in terms of the group duration.

For practical purposes, it is likely that origination of sensitive documents would be carried out on a PC, as much for the practicality of typing as for any security reasons. Therefore we are aware that one or more desktop client applications would be required for any fully-implemented solution. For the purposes of this "proof of concept" project, it was decided that an outline version would be created in Java in order to verify the cryptography, but that time would not permit this to be developed and tested to a level where it could be submitted as part of the finished product.

As discussed in section 3.4 the decision was made to use digital signatures as part of the security design. This generates the requirement for every group member to have a certificate that can be used for authentication. It is left to the discretion of each group to determine whether this should be issued from a recognised Certificate Authority or whether self-signed certificates are to be allowed. (For small groups, the process of manually confirming the certificate fingerprint by telephone may be acceptable, for example.) In order to contain the scope of this project it was necessary to assume that each user either had, or was able to generate a public/private key pair and the associated certificate.

Finally, it was assumed that the user acknowledges the sensitivity of his data and has taken reasonable steps to protect it, including the deployment of a device locking code and strong passwords.

Threat and Trust Model

For the purposes of this project, the following assumption regarding the threat and trust model were made:-

1. a group member may consider all other group members as trusted.
2. the Certificate Authority may be considered as trusted.
3. all data in transit or at rest on a third-party server is considered to be exposed to an attacker.
4. data at rest on the external storage of the device should always be considered as vulnerable to attack
5. data at rest within the application's internal storage is protected to some extent by the platform's in-built security mechanisms, although this should still be encrypted to protect against a casual attacker who is able to gain root access for the purposes of snooping in storage. The prototype is not required to implement software protection against a more sophisticated attacker, however we are aware that the use of hardware-based cryptography via a Trusted Platform Module (TPM) would ameliorate the effect of this type of attack.

3.3 Document Storage

When considering how to manage the storage and sharing of group documents, we initially considered implementing a custom web server. However, it was recognised that this option had a significant IT overhead, both in development and ongoing support and therefore did not fit with our requirements. It was deemed that a better solution would be to leverage the facilities already available via one of the widely used cloud storage services: Dropbox, Google Drive and OneDrive were considered.

Although in a production-level application it would be desirable to allow the user to choose which cloud storage service they wished to use, it quickly became apparent that to implement using multiple APIs was infeasible within the scope of this project. However, as the facilities offered by each were broadly similar, it was felt that developing for a single API in the prototype would be sufficient for the purposes of validating the concept. Dropbox was selected as it is widely used and is well supported in both Windows and Linux environments, the latter being important as this was the development and testing environment being used for the project.

As described in 2.2, Dropbox offers several version of the API for Android and the Sync API seemed to be the best match for the project requirements. Developers are encouraged to assign each application the least permissions possible, so initially App level permission was chosen. However, it came to light that this did not allow any access to shared folders

so the design specification was amended to use the File type permission. This presents a filtered view of the user's Dropbox folder, only permitting the calling application to see files of the specified types. One disadvantage of this is that although all of the user's folders can be seen, various folder operations (e.g. delete, share) are prohibited as the folders may contain hidden files.

When registering SecurelyShare, text and document types were required but it was also necessary to have a file extension to represent our encrypted files. It would have been possible to submit a request to register custom file types with Dropbox but since time was of the essence, we elected to choose two less-commonly used ebook formats as a temporary measure, .xps (for encrypted documents) and .xeb (for encrypted group keys). The .xps extension was simply added to the filename of an encrypted file so that it was easy to recover the original file extension on decryption.

Administration

Although Dropbox provides an excellent environment for collaborative working with its inbuilt mechanisms for file and folder sharing, this does bring with it the initial administrative overhead of creating a group folder and sharing it with all the members. As discussed, the Dropbox Sync API does not currently provide support for this in Android but it was deemed acceptable for our purposes that this process should be completed via the Dropbox website in the usual way.

3.4 Cryptographic Design

There are two main aspects to the cryptographic design as follows:

- the document encryption scheme
- the group key distribution protocol

Document Encryption

As one of the central assumptions is that any third-party server cannot be trusted, it is essential that client-side encryption is used to protect our sensitive data. SecurelyShare uses a symmetric key encryption scheme and is currently implemented using the AES algorithm in Cipher Block Chaining (CBC) mode with a key length of 256 bits. A design requirement

was that references to algorithms and key lengths should be included as static variables rather than being hard-coded in order to provide an easy upgrade path or scope to make these user options at a later date.

Since a user of SecurelyShare may be a member of multiple groups at any one time, it is important that the decryption module is able to determine the correct key to use for each document. Initially we thought to capture this information based on the name of the containing folder but quickly realised that this was prone to error as, if a file became disassociated from its correct folder there would be no way of determining where it came from and hence reuniting it with the correct key. It was determined that each file should have a file header written to it made up of an integer containing the length (in bytes) of the group name, a string containing the group name and a fixed length Initialisation Vector - this would provide all the information required to decrypt the file.

Group Key Distribution

By their nature, symmetric schemes require that all parties are in possession of the key, as the same key is used for both encryption and decryption. Where we are just sharing a key between two parties, methods such as the Diffie-Hellman key exchange protocol [?] are widely used. However, the problem of group key distribution is much more challenging and has been the subject of multiple research papers (see 2.3). Although these papers present a number of interesting theoretical schemes with advanced features, our design goal was to produce a practical but secure implementation which matches requirements built with well-established cryptographic primitives and protocols.

Before generating the group key, a Java KeyStore should be set up containing the private key and corresponding certificate which is to be used to authenticate the group key on distribution. Each group member should supply a certificate which should then be imported into the KeyStore and finally the key generation module can be run.

figure should go here with a flowchart for this but will just include description for now

A high-level description of the key-generation process is as follows:-

- generate a new symmetric key for the group
- for each user certificate in the KeyStore, encrypt a copy of the group key with the user's public key (as contained in the certificate) using RSA
- sign the encrypted key using the administrator's certificate

- save the encrypted and signed key to the group Dropbox folder

The use of digital signatures is required for each user to be able to validate that the encrypted key they receive is the authentic one and not one substituted by an attacker. This is a vital part of the security protocol and hence has been included in our system design. However, as will be discussed in Chapter 8, due to some of the challenges encountered during the implementation phase, we were not able to implement this aspect of our design fully in the prototype.

Should I discuss choice of PKI over ID based scheme here or in evaluation

3.5 Android Prototype

Encryption

As outlined in the Problem Analysis (3.2), it is unlikely that large documents will be originated from within SecurelyShare so our design just included a simple edit window for text input which can then be saved to Dropbox as an encrypted file. For completeness, and to aid testing, we also included the ability to encrypt a file which already exists on the device (although this is not to be encouraged as such files may have been exposed to an attacker prior to encryption).

Decryption

Since tablet devices are ideally suited for reading documents, the major requirement of this app is that it is able to take encrypted documents stored in the user's Dropbox, decrypt them with the appropriate symmetric key and present the original text to the user in a readable format. Our requirements necessitated that this should be accomplished without the need to save the plaintext to disk on the device. Initially this seemed as though it would be achievable as the Android platform provides mechanisms for one application to provide data to another,. However, on deeper study, it transpired that none of these options was exactly right for our purpose. We considered a number of ways of modifying our design in order to overcome this challenge but all of them required some sort of modification to the third-party application. Eventually it was decided that we had to weaken our requirements or risk jeopardising the entire project time scale.

It was agreed that we could achieve our objective for simple text files by displaying the decrypted data in one of the Android-supplied Edittext widgets, which would enable it to be read and, if required, edited saved again in encrypted format. For files requiring a more complex viewing and editing environment, for example those in Portable Document Format (PDF), we would write these temporarily to the external cache before broadcasting an Intent with the appropriate URI in order to allow the user to open this temporary file with one of the PDF readers installed on their device. The security implications of this and suggestions for future work are discussed further in Chapter ??.

Key Management

When first installed on the Android device, the SecurelyShare application will create two keystores and save them in the application's private storage. The Certificate keystore contains a copy of the user's private key and corresponding certificate chain and the Group keystore is the place where all the group keys will be stored. KeyStores are mechanisms designed for securing cryptographic artifacts, encrypted using Password Based Encryption (PBE) with a user supplied password.

As described above, the group key is created by the group administrator and a file is generated for each group member containing a copy of the group key encrypted with the private key contained in the Certificate KeyStore. Before the user can begin creating or reading documents for the group, this key must be imported into the Group keystore on the device. The user selects his file from the Dropbox folder containing the encryption keys for the group, the app decrypts this with the user's private key and stores it in the Group keystore using the group name as an alias.

Chapter 4

Implementation

4.1 Development Environment

An early challenge encountered in this project was setting up a suitable development environment. Initially the intention was to use the Android Development Tools plug-in for Eclipse, as this was an extension to an already familiar IDE, however, difficulties were encountered with the school computers and finally we elected to switch to Android Studio (Beta) as suggested on the Android Developers website ?.

Since the main rationale for the Android application was to allow users to read documents away from the PC, we felt that the application should be optimised for use on a tablet. As difficulty was encountered getting the device emulators to work on Linux, it became necessary to purchase a physical device for the purposes of the project. As budget was a significant consideration, the device chosen was a Hudl 7" tablet running the Android 4.2.2 Jelly Bean operating system. For the purposes of this project, it was decided that we would develop and test for this device. This meant that we were restricted to API 17, which eliminated the need to include any backward compatibility but also precluded using features of the latest Android releases as it transpired that there was no upgrade path available for this device. This fact turned out to be highly significant in a later stage of the project.

4.2 Initial Development

As we had no previous experience of developing for the Android platform, it was felt wise to develop a very simple application in order to gain some familiarity with the complexities of the Android architecture and its API. This initial application enabled us to input some text, encrypt it (using a simple "convert to upper case" as a place-holder for encryption) and write it to the application storage.

The next stage was to introduce real encryption, for which we needed to investigate the Java Cryptography Architecture(JCA). Although the choice of algorithm was determined as part of the design, we were mindful of the fact that many security weaknesses are introduced by poor implementation. As Schneier (1998) states,

And just as it's possible to build a weak structure using strong materials, it's possible to build a weak cryptographic system using strong algorithms and protocols

For this reason we decided that we would initially develop and test the Java classes to be used for encryption and decryption in the more familiar Linux environment before porting to Android. Linux also provides a simple command line interface which was valuable for inspecting the binary (or hex) content of encrypted files to aid debugging.

/red I am still tidying up from here on - have just left headers as placeholders for now

4.3 SecurelyShare Prototype

In order to widen our experience of Android development, we sought to include a variety of different techniques

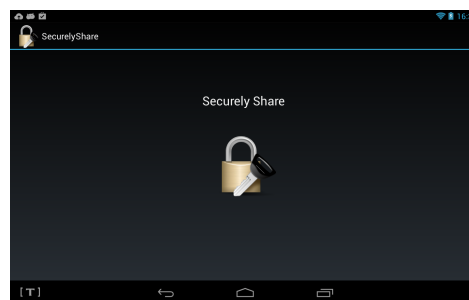


Figure 4.1: Initial splash screen

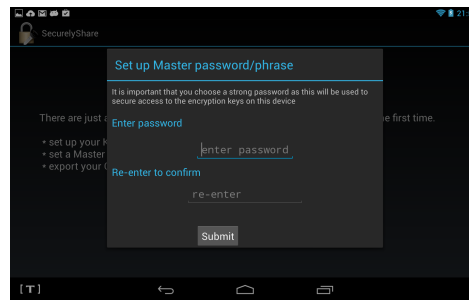


Figure 4.2: Setup Master Password

4.4 Implementation Challenges

4.5 Testing

Chapter 5

Project Management

Chapter 6

Evaluation

6.1 Security Evaluation

6.2 Product Evaluation

6.3 Process Evaluation

As outlined in the Introduction, one of our goals in yundertaking this projhect was to gain some experience of development for the Android platform and as such it was important that we aspired to seek out and use 'best practice' wherever possible.

6.4 Future Work

Chapter 7

Conclusion

Chapter 8

Implementation

8.1 Development Environment

An early challenge encountered in this project was setting up a suitable development environment. Initially the intention was to use the Android Development Tools plug-in for Eclipse, as this was an extension to an already familiar IDE, however, difficulties were encountered with the school computers and finally we elected to switch to Android Studio (Beta) as suggested on the Android Developers website ?.

Since the main rationale for the Android application was to allow users to read documents away from the PC, we felt that the application should be optimised for use on a tablet. As difficulty was encountered getting the device emulators to work on Linux, it became necessary to purchase a physical device for the purposes of the project. As budget was a significant consideration, the device chosen was a Hudl 7" tablet running the Android 4.2.2 Jelly Bean operating system. For the purposes of this project, it was decided that we would develop and test for this device. This meant that we were restricted to API 17, which eliminated the need to include any backward compatibility but also precluded using features of the latest Android releases as it transpired that there was no upgrade path available for this device. This fact turned out to be highly significant in a later stage of the project.

8.2 Initial Development

As we had no previous experience of developing for the Android platform, it was felt wise to develop a very simple application in order to gain some familiarity with the complexities of the Android architecture and its API. This initial application enabled us to input some text, encrypt it (using a simple "convert to upper case" as a place-holder for encryption) and write it to the application storage.

The next stage was to introduce real encryption, for which we needed to investigate the Java Cryptography Architecture(JCA). Although the choice of algorithm was determined as part of the design, we were mindful of the fact that many security weaknesses are introduced by poor implementation. As Schneier (1998) states,

'And just as it's possible to build a weak structure using strong materials, it's possible to build a weak cryptographic system using strong algorithms and protocols'

For this reason we decided that we would initially develop and test the Java classes to be used for encryption and decryption in the more familiar Linux environment before porting to Android. Linux also provides a simple command line interface which was valuable for inspecting the binary (or hex) content of encrypted files to aid debugging.

DROPBOX For improvement, use custom file extension registered with Dropbox then would only ever see encrypted files

GOOD PRACTICE use of bundle for passing data between activities

use of interface for passing data back from dialog

Use of singleton

ANDROID FEATURES

Splash screen and initialization

Fragments

Implemented with algorithms as static variables so that easy to amend if standards change

CRYPTOGRAPHY

CHALLENGES Major issues with keystore, certificates and default providers.

dbx stuff does not implement serialisable or parcelable

Challenge of unavailability of BKS on pcs in school

No access to key tool in android

Issue: if generate [private key on device, it is device specific - ability to import would allow same keystores to be used on multiple devices for4 same user

Issue: if generate [rivate key on device, it is device specific - ability to import would allow same keystores to be used on multiple devices for4 same user

THINGS NOT INCLUDED However, as will be discussed in Chapter 8, due to some of the challenges encountered during the implementation phase, we were not able to implement this aspect of our design fully in the prototype. (digital signatures)

custom pdf reader

OPTIMIZATION

buffering

decision not to implement threads at this stage Did not optimize for battery and memory

Admin system – basic with no front end. Designed to run on system with Dropbox installed and running so files uploaded to dropbox simply by saving them in the correct location rather than worrying about using Dropbox API within program.

Android prototype

PC version in java that just does basic encryption/decryption. Used primarily for intial development and testing of encryption mechanisms. Single username hardcoded for testing. Fully developed application with user interface outside of scope of this project. Assumption that encrypted blobs are probably also created on PC - simple PC version of program developed to address this, although no gui developed

8.3 Testing

The encryption and decryption elements of our prototype were largely testing by inspection - known texts were encrypted, the files were checked to ensure that they were not leaking any of the plaintext, and these were in turn decrypted and compared with the original. In general, the nature of encryption means that the main body of a file either decrypts completely or not at all, however particular attention must be paid to the very beginning

and end of a file, as incorrect use of the API can lead to errors here. As our project was aimed at document encryption, we tested using a mixture of text (.txt files) and PDFs (although we did also confirm that our processes generalised to other file types). Since randomness plays a significant role in ensuring semantic security for symmetric encryption schemes, this made any sort of unit testing extremely difficult. At the outset we considered using fixed initialisation vectors and encryption keys to make our tests repeatable. However, on closer consideration this involved making some significant changes to the program logic relating to the use of the cryptography APIs and so was considered to make this sort of testing meaningless.

Other aspects of the user interface were tested manually, trying to make sure that a range of user inputs and behaviour were covered. However, it should be noted that the prototype was not designed to be releasable code and therefore did not include comprehensive validation of user input. Although certain common scenarios were handled (for example, the user is informed if trying to decrypt a document without first importing the appropriate group key), it was deemed that for the purposes of this prototype it was acceptable that other less common behaviours should be permitted to cause the app to terminate (e.g. if the app permission is revoked via the Dropbox website whilst the app is running).

When a user shifts focus away from the current activity, the activity is kept in memory but is switched into the background. Such processes can be killed at any time by the system in order to reclaim memory. In this event, when the user returns to the activity, it is recreated by the system - this may cause some expected results if not designed for properly. Since this process is managed by the operating system and is determined by a number of external factors, it can prove difficult to test. If testing in an emulated environment, it is possible to simulate an artificially small device memory to encourage this behaviour to be triggered. However, as previously mentioned, this option was not available to us so a similar effect was created by rotating the device, which also causes an activity to be recreated. Although not identical, it proved quite an effective substitute in our testing.

Chapter 9

Project Management

In managing this project we used an iterative approach based on the Agile philosophy, although with such a small development team and no external customer many of the processes and artefacts associated with Agile methodologies were inappropriate. In particular, although we had fully working software at the end of each iteration, it was questionable if it could have been considered a 'potentially shippable product' (Gupta, 2008).

At the outset we had very little knowledge of the Android platform which resulted in it being difficult to form an accurate plan. Although Android applications are generally programmed in Java with which we had some familiarity, we had underestimated how much developing for mobile devices differed from anything we had previously done. We realised that a greater portion of the overall development time needed to be given over to production of the prototype, particularly because one of our initial aims was that we would seek to understand and implement best practice in our code. As a result we had to readjust some of the timescales in our project plan, however this was recognised early and was able to be well managed.

One challenge of using an iterative development model was that sometimes time was wasted on learning and development that was critical to get a particular iteration to work but which ultimately ended up being abandoned or didn't contribute significantly to the final product. It was important to guard against retaining features in the final design simply because of the amount of work that had gone into implementing them in an earlier iteration.

During development some elements which were required for security came to light. Some of these were implemented in the prototype, for example we delete the KeyStore after three failed password attempts, in order to protect against a device which had fallen into the wrong hands, whereas others were merely noted in the security design. Also, as discussed

in Chapter 8 we discovered a difficulty with opening decrypted pdfs without the need to write them out to a temporary file which made it necessary for us to revise the specification of the prototype.

One of the greatest challenges encountered was what is frequently described as "scope creep". The role of the prototype as a proof of concept was clearly defined at the outset and therefore the design goal that functionality should take priority over any usability considerations.

Regular contact with supervisor was extremely <http://www.scrumguides.org/scrum-guide.html#artifacts-increment>

Bibliography

Das, A., Bonneau, J., Caesar, M., Borisov, N., and Wang, X. (2014). The tangled web of password reuse. In *Proceedings of NDSS*.

Gupta, M. (2008). Definition of done: A reference - scrum alliance.

Schneier, B. (1998). Security pitfalls in cryptography. In *EDI FORUM-OAK PARK-*, volume 11, pages 65–69. THE EDI GROUP, LTD.