Table of Contents

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-drivent 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 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 4 and 5 provides details of the implementation, testing and management of the project, Chapter 6 reviews the security

features and finally Chapter 7 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 an 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 applicable:

- 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.
- 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.)
- Full Dropbox: the app is given unrestricted access to the user's Dropbox

2.3 Review of Related Work

need to include a summary of what is out there relating to group key distribution

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 organisations concerned would have no access to decryption keys. Although there are a significant number of encryption applications available, our research was not able to identify any robust security comparison of the various products on offer. We decided to consider two commercially produced applications, Boxcryptor and SafeMonk.When trying to evaluate these products there appeared to be a paucity of robust reviews from respected members of the security fraternity and many of the review that were available seemed to focus more on the usability of applications rather than on their security features. One of the only ways that a security application can truly be evaluated is if the source code is available for community scrutiny as security flaws often fail to come to light just from using the product. Most of the products we looked at seemed to be designed primarily for personal file protect - some professed to offer sharing although some of the less polished versions required manual sharing of encryption keys. Some of them were unforthcoming about exactly how they work

Two products that we examined in greater details, partly because they seemedto be higher profile products although that may simply be due to the fact that they were commercial products with a paid version as well as the free version that we tested and hence had larger

marketing budgets. It may also be because they had more security credibility however there was a common trend among the examined applications to use AES for encryption of the actual files themselves and some implementation of RSA to manage the key exchange, In all the cases that we examined there was a central server that was used for key management, although both Boxcryptor and Safemonk were keen to strees that they were "tapproof" or "zero-knowledge" servers and had no access to key material that would enable them to decrypt user files. 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)

"http://source.android.com/devices/tech/security/system-and-kernel-level-security"

"http://link.springer.com/book/10.1007/978-1-4302-4063-1page-1"

Chapter 3

Analysis and Design

In this chapter we will present some 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.

3.4 Cryptographic Design

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

- the document encryption scheme
- the key distribution protocol

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.

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)

Include information about algorithms and key lengths File meta data

Reasons why didn't choose ID based cryptography or password based solution - aim is to use simplest solution that works. Refer to examples which use PBE. Happy to use it for protection of key material on device as even with password, attacker would have to have access to device or would require more sophisticated remote attack which is out of our scope.

3.5 Group 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 the Dropbox API does not currently provide support for this in Android, it was deemed acceptable that this process should be completed via the Dropbox website in the usual way.

include screen shot of folder structure

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)
- 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 4, 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.

3.6 Android Prototype

I would develop prototype for a single API and test on one device – no backwards compatibility hence no need to use compatibility libraries eg for Action Bar

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

can open a viewer via an Intent but then required to return a link to a file. Considered offering decryption as a Service, but this would still require modification of the calling app in order to persuade it to request decryption. A number of ways of modifying our design in order to overcome this challenge but eventually it was decided that we had to weaken our requirements or risk jeopardising the entire project time scale. Consequently we decided 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 and resaved in encrypted format. For files requiring a more complex viewing and editing environment, for example those in Portable Document Format (PDF), we currently have 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 and requirements for future work are discussed further in Chapter 6.

Key Management

No ability to recover keys if app gets deleted - could re download from Dropbox as long as private key and certificate backed up.

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, difficukties were encoulntered 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 fro 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.

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 PERFORMANCE Performance problem - introduced buffering

ANDROID FEATURES use of xml rather than java for managing onClick - why was this done and when is it not applicable Splash screen and initialization Fragments Implemented with algorithms as static variables so that easy to amend if standards change CRYPTOGRAPHY

Removal of keystores upon 3 successive failed password attempts – al present it just shows a message saying that the keystores have been deleted but doesn't actually remove them from the device. In a live system this would need to be implemented. CHALLENGES Major issues with keystore, certificates and default providers. dbx stuff does not implement serialisable or parcelable Dropbox synchronization issues - developed everything using App specific access then discovered that this doesn't allow any use of shared folders so had to redesign 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

how to manage letting decryption know group - flirted with shared preferences 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 4, 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

4.2 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 on 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 possibly 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 5

Project Management

We used an iterative approach, loosely based on Agile methodologies however with such a small development team and no external customer it was deemed inappropriate to use all of the porocesses and artifacts of any single agile methodology. In particular although we strived to maintain working software at the end of each iteration this was not the production quality software which would be required in a true agile model where one is expected to deliver minimum viable product at the end of each iteration.

At the start of the project we knew very little about the android platform - we were starting from a zero-knowledge base and therefore more time was taken than anticipated in learning the android development environment, particularly with the proviso that in our initial aims we had set that we would seek to use best practice in protoype code. During developent it becme apparent that timescales would have to be revised and this was duly done and well managed.

During development some elements which were required for security came to light. Some of these were implemented in the prototype whereas others were merely noted in the security design. There were parallel developments - the security design was improved over time as the knowledge base was increased. The other one was the prototype where on occasion we had to revise what was actually going to be achieveable within the time fram and some elements got moved from within the prototype into the design spec so that they became part of the security design but were not actually implemented. For example the proviso of never writing plaintext to disc as is discussed in xxx was a requiremnt that had to be removed from the initial protoype deliverable.

Some significan challenges were encountered at the beginning to do with actually getting

our development environment set up on the school computers. This again impacted on the timescale as it exceeded the original margin allowed.

Version control was managed using a github repository.

No actual time was scheduled per task as there was no team of people to be organised.

One of the challenges of using an iterative development model was that since there was a lack of experience with the Android platform at the outset it was difficultto form a accurate pland and on occasion time was wasted on work that was critical to get a particular iteration to work but ultimately ended up being abandoned or didnt contribute significantly to the overall fiished product. There was a danger of keeping things in the final design because of the amount of work that had actually gone into getting them working in the first place and this temptation was one that presented a challenge to overcome

Challenges of Agile methodology when using a new language or API - lot of time spent learning how to implement stuff that ended up not being needed. Use of Java program running on PC to develop encryption to begin with e.g. shared preferences - implemented during one of the iterations but ultimately abandoned for a simpler model keystores may have been handled differently once had to use Bouncy Castle anyway

`http://www.scrumguides.org/scrum-guide.html#artifacts-increment`

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.

Chapter 6

Security

The issue with our solution scalability is just the admin cost of setting up the dropbox folder, adding all the users and getting all the certificate and importing into the keystore in order to run the initial key generation. In our solution it is possiblethat this function could be delegated tosomeone who isn't part of the trusted group and they would never have any access to the shared secret. However it is important that they are honest at the initial setup phase or it would be possible for them to insert Mallory's certificate into the keystore and hence they would generate a copy of the encryption key for Mallory. However, as long as they are honest at the outset, there would be no opportunity for Mallory to subvert them at a later stage as being able to get access to the keystore containing all the certificates wouln't give any advantage. Gaining access to the Dropbox forlder would enable Mallory to see what files were in there but without private keys he woulnt be able to decrypt anything.

With our scheme, if delegating the admin function it is important that the membership of the group is known at the outset as the encryption key is ephemeral, however it could be the subject of further work to extend the android application to permit the Android application to include the capacity to share the group key with a new user. This would need to be considered carefully as it would provide an additional means of Mallory getting access to the key, even if he only had temporary access to the device whilst the application was unlocked.

There is currently no means of revoking a key, however the fact that we are also making use of the access control afforded by Dropbox would also give some protection in the event that a member left the group as their access to the Dbx folder could be revoked. This is one of the reasons why we suggest our solution my not be scalable to large groups as in

this instance the membership is much more likely to be volatile. It could be the subject of further work consider alternative key sharing mechanisms to see whether improvement could be achieved here without introducing the requirement for a central key management server.

Forward secrecy - requiring perfect forward secrecy adds an additional level of complexity to any encryption. The main application of forward secrecy to our project is that a member who leaves the group should not have access to future documents.have I got this right? Although our current cryptographic design does not provide this (once established, the group key is never changed), one of the benefits of using a service like Dropbox is that we are able to avail ourselves of its extensive access control mechanisms. In order to compromise security an attacker needs access to both the group's shared Dropbox folder *and* the group key - a member leaving the group would only have access to the former, which is considered sufficient for our purposes at this point. An evaluation of more sophisticated key exchange protocols in order to strengthen this aspect of security could be the subject of further work.

Client side encryption was used because of the desire to ensure that server had no access to plaintext

digital signatures to protect against man in the middle is it really MITM

Dropbox uses SSL so protected agains replay attacks

Had to compromise on aim never to write to storage - this is a temporary measure but would need to implememnt a custom pdf reader which is beyond the scope of this project. It is interesting to note that this is the same issue experienced by xxx application

Write about why I didnt use encrypted folder approach

specifically designed to avoid central server for key management. Essential to avoid running our own and placing trust in 3rd party. Our solution is similar to commercial ones - they make much of being zero-knowledge but't essentially this is a matter of trust as can't examine their code.

Problem with API meant that we had to generate keystore for private key and corresponding certificate chain manually. Further work would be to redesign this part of the app using the additional ffeatures offered in Android 4.3. Currently consider this to be one of the weakest parts of the design - by the time I realised that this could have been done better, it was too late to go back and change it.

The security implications and requirements for future work are discussed further in Chapter 6.

Password policy - balance of requiring password for every file access which encourages user to choose insecure psasword. Opted to use one password both to unlock the keystore and to encrypt each of the group keys. Could have had a separate password for each group but requiring user to remember different passwords often leads to insecure ones. xxx app offers different levels of password policy as a user option - this may be a useful addition to be considered in the future. Similarly, for ease of testing and general usability we decided not to implement encryption of filenames in the prototype, However, this would again be a potential usre option.

loss of device - we need to include some protection against loss of device but an attacker who is able to gain access to the device whilst the app is running would be able to use the app to decrypt any documents in Dropbox as key store would be unlocked. It should be noted however that the use of Dropbox does provide some additional protection for this as, once aware that the device is lost, the user is able to log in via the Dropbox website and revoke SecurelyShare's permissions for the account with immediate effect.

Backward secrecy - this is often required in a messaging environment where it is important that new group members are not able to decrypt messages from before they joined the group. However, in our application backward secrecy is not required as all group members should have access to historic documents.

Removal of keystore after password failure

Chapter 7

Evaluation

ACHIEVEMENTS • What works well EXPLAIN HOW WELL SOLUTION MEETS OBJECTIVES - WHAT YOU HAVE LEARNED - WHY ANDROID DEVELOPMENT WAS A CHALLENGE major challenge of the fact that android is an operating system not a programming language - event driven programming FURTHER WORK • From prototype to production - next steps Write as though you are porviding a basisi for a good cs graduate to continue the work - assume they have already done some android development SECURITY EVALUATION Did not implement signing in prototype as largely meaningless with self-signed certificates. Purchase of appropriate certificates for authentication of signatures would be required for a complete solution Attacks and issues to consider • anonymity • forward secrecy • revokation • man-in-middle Delete keys after failed password attempts Write about how p[rotocols as as important as implementation - need to support this view from academic papers Talk about why it doesn't matter that encrypted copies of group key are available on dropbox nelenkov.blogspot.co.uk - credential storage enhancements in Android 4.3 out of bounds channel - side channel attack Write about issues to do with public key distribution and the need for signing Talk about decision not to implemenmt passing decrypted data directly to another app without needing to write to external storage Don't zero out passwords after use No implementation of digital signatures so vulnerable to man-in-middle Decision to use same password for keystore and aliases - trade off of added security against temptation for users to use insecure passwords or write them down

Group needs admin, although any group member can serve in this role. It is also possible to delegate this to an administrator who is not part of the group without giving them access to the group encryption key. However, if the admin was corrupt, the fact that they had access to the private key for signing the encrypted group key would still be a problem..

Useful phrase "a more sophisticated attacker"

Threats: • Malware on device • Attacker snooping around external storage but not one with root access • Lost device with app open (minimal protection) but can unlink from dropbox remotely so would only have a very small window of opportunity to decrypt files currently stored on devicewhilst keystore is unlocked • Could have had different password for each group • Could make user re-enter password for each file – trade off between added security in event of lost device and templtation for user to choose a weaker password

Maybe argue why solution is secure here PROTOTYPE EVALUATION dependent on exactly correct alias for groupid and folder name • Is designed as a "proof of concept" • Aspires to use "best practice" within the code • Uses well-tested cryptographic techniques and standard libraries • Adheres to the stated security requirements No ability to change passwords etc added at present

EVALUATION OF PERSONAL LEARNING • zero knowledge starting point • Android is a whole new operating system not just 'Java with extra bits' • Unfamiliar API's operating in a sub-optimal environment

Since many of the example apps that we looked at were focused on personal encryption with the sharing capacity as an added feature, it was importatnt that we remembered that the core purpose of our application whas group collaboration and therefore central to any design we might chososse was the requirement for a robust key sharing mechanism.

scalability - Structure would work with larger group but there may be better key management protocols - admin issue with setting up dropbox shares etc.

Chapter 8

Conclusion

Bibliography

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

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