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 Securely Share, consisting of a detailed design of the security components of the system and a prototype android application (SecurelyShare) 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.

Cryptography Android provides a set of cryptographic APIs for use by applications.

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 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 [1] 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 abolutely no prior experience of developing for (or even using) an Android device, therefore this phase was highly iterative. As our familiarilty 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 materal in a single chapter is more reflective of the underlying process.

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

3.1 Problem Analysis

3.1.1 Assumptions

As outlined in the introduction, Securely Share is intended for use by small (typically of the order of 3-15 members), autonomous groups, with a largely static membership. This facor 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 deveoped and tested to a level where it could be submitted as part of the finished product.

As discussed in section **??** 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 confirmiing the certificate fingerprint by telephone may be acceptable, for example.)

A final assumption was 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.

3.1.2 Threat and Trust Model

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

- a group member may consider all other group members as trusted.
- the Certificate Authority may be considered as trusted.
- all data in transit or at rest on a third-party server is considered to be exposed to an attacker.
- the prototype Android application should aim to protect against an attacker accessing the device's external storage.
- where an attacker is able to gain root access to the device, a casual attacker would then be able to access the application's protected internal storage and the prototype should take measures to protect against this. The prototype is not required to implement software protection against a more sophisticated attacker with root access, however we are aware that the use of hardware-based cryptography via a Trusted PlatformModule (TPM) would ameliorate the effect of this type of attack

3.2 Document Storage

need to paste the stuff about why we implemented dropbox and not a custom server

Following initial analysis of the problem it was decided that the best solution, rather than trying to implement a custom server, would be to leverage the facilities already available via one of the widely used cloud storage services. Dropbox, Google Drive, OneDrive and Box were considered. Initially the plan was to allow the user to have the choice of which service they wished to use as a number of personal encryption apps have this facility. It quickly became apparent that accommodating all of the different APIs was going to be beyond the scope of our prototype and so we decided to select just one. We chose Dropbox because it was already familiar, it has good cross-platform support including a Linux client which was important as this was the development and testing environment being used. Although in the production level design it would be desirable to allow the user to choose which cloud storage service they wished to use, clearly to implement for multiple systems was infeasible within the scope of this project it was decided that a single API would be used for the purposes of the prototype as we were attempting proof of concept and this would be substantially the same whichever API was used.

Client server model was considered but has an IT overhead and , would need to ensure communications are secure. Could use server just to create and distribute keys but requires trusting server and needs user to authenticate to server Working on the basis that we wanted the simplest viable solution, looked at using existing cloud services that offer an android API

Intended to provide the user with a choice of which online file server to use but having begun the implementation using Dropbox, it quickly became apparent that attempting to design for multiple APIs was not realistic within the available timeframe, hence it was decided to develop the prototype with Dropbox initially as this is widely used and is well supported in both Windows and Linux environments.

3.3 Cryptograpic Design

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 od key material on device as even with password, attacker would have to have access to device or would requre more sophisticated remote attack which is out of our scope.

Early design considerations • who generates keys • how are keys generated • how are keys distributed • issue with public key - how would we stop Mallory uploading bogus documents • issue with where to place trust

3.4 Group Administration

manual processes as well as design of java application

Admin - could be done by group member or delegated to administrator. Relative merits of each approach and any modifications required to the code are discussed in Chapter 6

include details of manual admin tasks and screenshot of folder structure.

Folder management and sharing done outside of app - could be added but may require switching to a more complex dropbox api. Cost-benefit analysis

3.5 Prototype Design

Althou prototype allows for encryption of files stored on the device, in practice the very fact that there are files on the device that the user wants to encrypt violates our central tenet that plaintext should never be stored to disc.

• 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

This was a hybrid project so the design was just as important as the actual implementation. Since at the outset we had no experience of developing for the Android platform and little experience of developing security applications the design process was itself an iterative one

Chapter 4

Implementation

Implementation and testing. A detailed account of the implementation and testing of your software. Explain what data structures you used, and how the algorithms were implemented. What implementation decisions did you take, and why? There is no need to list every little function and procedure and explain its working in elaborate detail; use your judgement on what is appropriate to include. use of .xps, .xeb For improvement, use custom file extension registered with Dropbox then would only ever see encrypted files use of bundle for passing data between activities use of interface for passing data back from dialog Performance problem - introduced buffering Splash screen and initialization didn't use onStoreState etc. - didn't worry about restoring exact user position as prototype and system stores GUI stuff Use of singleton Keystores moved to external storage for testing and demonstration purposes. In a production app, these should be moved back to internal storage in order to take advantage of the additional protection afforded by android's inbuilt security mechanisms 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. • use of xml rather than java for managing onClick - why was this done and when is it not applicable Major issues with keystore, certificates and default providers. Challenge of absence of built in file manager DROPBOX ISSUES • 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 Include information about algorithms and key lengths Fragments

User authentication and need to block after failed attempts • large files • network connectivity • battery life • small memory • multithreading for gui • where to encrypt • model to use for file distribution and storage TESTING Testing - it is ok to say that I tested by inspection Explain why unit testing is not meaningful

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

Assumption that encrypted blobs are probably also created on PC - simple PC

version of program developed to address this, although no gui developed

There were a number of design decision points where it was possible to do something in the prototype, it was possible that the user may want to do it infrequently but it was perfectly possible to do it from elsewhere, for example there are perfectly good mechanisms for creating and sharing Dbx folder -and at present there is no means of sharing folders within the Sync API so there seemed little point in introducing the complexity of switching to a more complex API just for functionality that is easily done elsewhere and would only be required infrequently (it is really only a task required ast group setup and our view is that a group would be likely to have a lifetime of months

Use of Java program running on PC to develop encryption to begin with No ability to recover keys if app gets deleted - could re download from Dropbox as long as private key and certificate backed up.

3 parts

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.

Android has inherent protection by sandboxing apps so files in internal storage have extra protection. FOr the purposes of development and testing, files written to app protected external storage so that they can be inspected with a file manager without the need to root the device

decision not to implement threads at this stage

When app is deleted, files should be deleted so keystore would be removed from device

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

Write about choice of device

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

Chapter 6

Security

The issue with our solution scalability is just the admin tsat 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.

public wifi and know that data remains secure at all times

Encryption of filenames

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

digital signatures to protect agains man inthe middle

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

computational overhead acceptable

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.

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 chosose 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.

Bibliography

[1] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. The tangled web of password reuse. In *Proceedings of NDSS*, 2014.