Investigating the Security of Android Security Applications

Greig Paul
University of Strathclyde
Department of Electronic
& Electrical Engineering
Glasgow, United Kingdom
greig.paul@strath.ac.uk

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

***Abstract*—Encryption is commonly used to provide confidentiality of sensitive or personal information when held on smartphones. While many Android devices feature inbuilt full-disk encryption as a precaution against theft of a device, this is not available on all devices, and doesn't provide security against a device which is turned on and in use. For this reason, a wide variety of applications are available within the Google Play Store, offering to encrypt user data. Modern, strong encryption offers strong assurances of confidentiality when used correctly, although the fundamental cryptographic primitives are complex, with many opportunities for mistakes to be made.**

The security of a number of implementations of Android-based encryption applications is investigated. Highly popular applications, including those by Google-endorsed "Top Developers", are considered. A number of major weaknesses in the implementation of encryption within these applications is presented. This highlights the importance of both well-audited open-source cryptographic implementations, as well as the underlying cryptographic algorithms themselves, given the vulnerabilities identified in these applications. In many cases, there was no encryption in use by the application, and file headers were undergoing trivial static obfuscation, such that files would appear corrupted. In other cases, encryption algorithms were used, but with significant implementational errors. In these cases, plaintext recovery was still possible, due to the use of static keys for every installation of the app, and the re-use of cipher initialisation vectors.

I. Introduction

Encryption is a term many internet and mobile users are familiar with today; it is commonly accepted good practice to encrypt confidential data, and indeed many organisations' information security policies mandate the use of encryption for data security and confidentiality purposes [1]. This paper explores popular real-world implementations of encryption within a range of Android applications, and demonstrates that the mere presence of encryption is insufficient to ensure the security and confidentiality of user data.

Within the conventional model of security being defined as confidentiality, integrity and availability, encryption is used to provide confidentiality. It can also be used in certain modes of operation to provide integrity, where an cipher incorporating ciphertext authentication is used. Within the context of this work, however, applications will only be evaluated from the perspective of the confidetiality they offer for data. This is for because none of the applications investigated attempted to carry out ciphertext authentication.

This work was carried out through black-box analysis of a selection of Android applications, available on the Play Store, investigating the efficacy of various applications through analysis of the ciphertext outputs. All tests were initially carried out on a Moto G 2014 Android handset, with the Android Debug Bridge (ADB) interface used to gain access to the device's shared storage. It should be noted that third party applications can access any files stored here, through the widely-used `READ_EXTERNAL_STORAGE` permission, and could therefore exploit these weaknesses. Findings were then reproduced on a Nexus 5X handset, in order to verify that they applied across different devices. Wherever possible, all work was carried out with devices disconnected from the internet, to ensure that no external factors or automatic updates would affect the methodology used. The latest available version of each application was used when the work was carried out, and was downloaded from the Google Play Store. Unless otherwise stated, no modifications or root access was required on the device in order to carry out the procedures used.

Schneier's law dates back to 1998 [2], with his statement on the matter that anyone may design a security they themselves cannot break. This itself has roots in the work of Babbage in 1864 [3], where he stated "One of the most singular characteristics of the art of deciphering is the strong conviction possessed by every person, even moderately acquainted with it, that he is able to construct a cipher which nobody else can decipher." Despite this, much of today's widely-available cryptographic software continues to be built according to this premise, whereby it remains breakable. By considering widely-used encryption software on the Android platform, it will be demonstrated that users remain at risk due to fundamental misunderstandings of the use of encryption, or even what may be classed as encryption.

II. Previous Works

While a wide range of security and encryption applications are available on the Android platform, there appears to be little previous works considering their practical security. Previous

work considering the usability of security systems has considered PGP [4], and highlighted the risks to security posed by complex or otherwise confusing systems. In contrast, all of the encryption apps considered within this work were relatively user friendly. The security of networked Android applications has been widely considered, with various work exploring TLS implementations and validation of server certificates [5], [6], [7].

Egle *et al.* carried out a study of misuse of cryptographic constructs within Android applications in 2013 [8], which highlighted that 88% of applications they reviewed made errors in their use of cryptography APIs in Android. As part of this work, a number of rules were proposed, surrounding correct uses of cryptography within applications, including avoiding use of ECB mode, avoiding use of non-random IVs for CBC encryption, etc.

III. Applications Investigated

The applications listed in Table I were investigated. Note that "(TD)" denotes an application by a Google Play Top Developer.

Each application was installed on a test device, as detailed in Section I. For applications claiming to encrypt images, three JPG images were used for the test. The same set images were used for each application. For applications claiming to encrypt videos, a set of three MP4 videos were used. Methdology for the password manager considered is discussed in the relevant section.

IV. Hide Photo & Video Vault (vlocker)

The Vlocker application used package name `com.simpleapp.vlocker` [9], and version 1.0.1 was investigated, which was the latest version available, released on 20th January 2016. At the time of writing, it had between half a million and one million downloads from the Play Store, and 4574 reviews with an average score of 4.2/5 stars.

A. Developer Claims

The developer of the software claimed to use encryption within the application — "Vlocker is the Super Video Hider. Lightning encryption and password recovery feature allow your privacy more secure". More specifically, the developers claimed "Encryption - your videos are encrypted using advanced 128 bit AES encryption", although somewhat intriguingly also claim they allow for email-based recovery of a user's PIN within the application [9].

B. Operation of Application

The application was used to encrypt a short MPEG-4 video, recorded using the test handset. The original video was downloaded to the computer, prior to the encryption of the video. The application was then used to encrypt the video.

The file was no longer available at the path it was previously available. A hidden directory `.vlocker` was noted to have been added to the `/sdcard` shared storage area of the device.

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

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

Fig. 1. Comparison of original file and vlocker-protected file

Within this folder was the layout of the *vault* shown by the application. A new file was found, with the same name as previously, with the suffix `.vlocker` appended to it. This file was retrieved to the computer using ADB.

An initial inspection of the so-called encrypted file indicated that it had not been encrypted with AES-128, as claimed by the developers. Indeed, after the first 8192 bytes of the file, the remainder was byte-for-byte identical with the original file, indicating that only the MPEG4 header had been tampered.

Figure 1 shows a comparison of the original file with the file after processing by vlocker. Cursory inspection highlights that this header has not been encrypted by AES, as the output distribution is nonuniform. Indeed, where a byte of `0x00` would be expected in the original video, a byte of `0xFF` was seen in the vlocker-protected file.

Therefore, Equation 1a was formed, to determine the output of the encryption process. It can be reversed by re-arranging for decryption, as seen in Equation 1b.

$$ciphertext[i] = 255 - plaintext[i] \tag{1a}$$

$$plaintext[i] = 255 - ciphertext[i] \tag{1b}$$

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

C. Security Conclusions

Therefore, it can be concluded that, contrary to the claims of the developers, vlocker did not make use of AES-128 encryption. Indeed, it made no use of encryption at all — the protection applied to files is merely that from hiding them in a folder whose name begins with a "." character, which typically hides them from view, and by inverting the bytes of the MPEG-4 header.

V. Hide Pictures & Videos - FotoX

FotoX, available on the Play Store using package name `com.smsrobot.photox`, claimed that "all your private data will be secured, encrypted and invisible to other Gallery apps" [10]. The developers, SMSROBOT Ltd, are listed as being "Top Developers" on the Google Play Store, and FotoX

App Name	Package Name	Current User Count	Version Investigated
Hide Photo & Video Vault	com.simpleapp.vlocker [9]	0.5 to 1 million	1.0.1
Hide Pictures & Videos - FotoX	com.smsrobot.photox [10]	1 to 5 million (TD)	1.9
Vault - Hide Photos/App Lock	com.smsrobot.vault [11]	0.5 to 1 million (TD)	1.9
Photo Locker	com.handyapps.photoLocker [12]	10 to 50 million (TD)	1.2.1
Video Locker	com.handyapps.videolocker [13]	5 to 10 million (TD)	1.2.1
Password Locker - Data Vault	com.handyapps.passwordlocker [14]	0.1 to 0.5 million (TD)	1.0.2
Video Locker	net.newsoftwares.videolockeradvanced [15]	10,000 to 50,000	1.0.3
Gallery Vault	com.thinkyeah.galleryvault [16]	10 to 50 million	2.6.5
Encrypt File Free	com.acr.encryptfilefree [17]	50,000 to 100,000	1.0.8

TABLE I
DETAILS OF APPLICATIONS INVESTIGATED.

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

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

Fig. 2. Comparison of original file (upper) against FotoX-protected file (lower)

has between one and five million installs, with 21,131 reviews giving an average of 4.3/5 stars. Version 1.9 of the application was investigated, released in October 2015, the most recent available at the time of investigation.

A. Operation of Application

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

Within the directory `.FotoX` was the filesystem layout of the so-called *vault*, and these folders contained the protected files. The encrypted file had the suffix `.quickcrypt` appended to its name. It was extracted using ADB, and compared to the original file. This comparison indicated that only the very first few bytes of the file differed. Figure 2 shows a comparison of this region of the header of the file.

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

B. Security Conclusions

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

C. Similar Applications

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

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

VI. HANDYAPPS (VARIOUS APPLICATIONS)

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

A. Developer Claims

The developers of Video Locker claimed it is "the ultimate secret gallery app", and that a key feature is its encryption:

> Encryption - hidden videos are not only moved to a secret location on your phone but are also encrypted using advanced 128 bit AES encryption. This means that even if someone manage to steal your SD card and copy the hidden video files, they will still be unable to view the locked videos [13]

Identically worded claims were made for Photo Locker [12], including the assertion of 128-bit AES encryption. Since the operation of the two applications were found to be near identical, the operation of the two shall be considered together — only minor differences exist between the applications, specifically around the types of file accepted by each application.

B. Operation of Application

Upon encrypting a video with Video Locker, it was removed from its original location. A new folder within the shared storage, named `.VL` was identified, containing the vault contents.

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

Fig. 3. Comparison of original file (upper) against Video Locker-protected file (lower)

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

Fig. 4. Comparison of two different Video Locker-protected files

Each file had `.vl` appended to its name. Photo Locker created a similar directory named `.PL`.

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

Figure 3 shows a comparison of the first 144 bytes of an original video file, against the protected file, indicating the apparently-encrypted data.

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

The header of the two encrypted files were now compared — if these two ciphertexts held similarities, there would therefore be a correlation between the ciphertexts, indicating that similar plaintexts revealed similar ciphertexts. As shown in Figure 4, the two headers were similar for the vast majority of bytes, indicating that this was likely use of a statically initialised cipher. Indeed, the offsets of the bytes differing in the ciphertext were the same offsets as the bytes differing in the plaintexts. For example, at offset `0x1a` of Figure 4, two bytes differ between the ciphertexts.

Figure 5 shows that these same bytes in the plaintext

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

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

Fig. 5. Comparison of the original video files

differed. For ease of comparison, those bytes are highlighted with arrows.

Therefore, it was clear that, given the correlation between the two ciphertexts, the same key and initialisation parameters were being used, even though a different PIN was being used in the application for the encryption of the second file. This was confirmed by using the XOR function across differing bytes of both plaintext and ciphertext. For example, from Figure 3, bytes `0x31 - 0x33` were $[03, 20, 8A]$ in VID_3, and $[00, E7, D9]$ in VID_4. Within the corresponding ciphertexts in Figure 4, these bytes were $[10, BD, 03]$ and $[13, 7A, 50]$ respectively.

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

Since the same key was used for all data encrypted by the app, and this was static between different installations of the app, this makes the ciphertext vulnerable to a simple known-plaintext attack. By encrypting a large file, and recording both the plaintext and ciphertext of that file, any arbitrary file may be decrypted with the XOR function, since an unknown ciphertext was able to be XOR'd with the ciphertext whose corresponding plaintext is known, and the result XOR'd with the known plaintext to reveal the unknown plaintext.

C. Decryption of Video Locker & Photo Locker Data

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

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

— following decoding from base64, the number of bytes was the same as the length of each string.

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

D. Password Locker

Password Locker is an application by Handy Apps, the same developer as Video Locker and Photo Locker. It has between 100,000 and 500,000 users, and an average rating of 3.9/5 from 1,179 reviews. Password Locker, as the name suggests, is designed for the secure storage of passwords by users, and makes a large number of security claims.

For example, the developers stated that Password Locker "stores your sensitive information offline and passwords safe, secure and organised", and that it has "many optional convenient features for it to be the best password manager ever designed specifically for Android" [14]. Specific details were also given as to the encryption supposedly used — the developers stated:

> *Secure*
> Lock up your data in Password Locker with extremely tough and strong 256-bit AES encryption - military level encryption (takes trillions of years to decrypt) [14]

Password Locker stores its password database within the application's private storage, meaning that root access is required to retrieve the database. Note, however, that it also offers a paid feature to enable cloud synchronisation of passwords with a user's Dropbox or Google Drive account, which may expose this database to third party services. Nonetheless, given the ease with which Android devices may be rooted with exploits such as CVE-2014-3153 (TowelRoot) and CVE-2015-3636 (Ping Sockets root), we consider it viable that a malicious party may easily gain access to the database. Nonetheless, if the claims made by the developers are accurate, users would have nothing to fear, as their data would be appropriately encrypted.

The database was found to hold base64-encoded fields, in the structure of the records, as shown in Figure 6. From this, it was immediately possible to identify the use of weak, potentially broken cryptography, by the observation of a common prefix between the two *ciphertexts* for `bank_label` and `acc_name`. Since this was simply the default record created by the application, it was possible to verify the hypothesis that they shared a prefix of 3 characters (given the 4 base64-encoded characters in common). Indeed, this suspicion was found correct — the bank account label was "Sam Sample" and the account name was "Sample Checking Acct". Once again, there was no padding present in the ciphertexts, and

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

Fig. 6. Password Locker Database Structure

lengths of plaintexts could be identified directly from ciphertext lengths, due to a lack of padding.

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

$$key = kc \oplus kp \tag{2a}$$

$$up = key \oplus uc \tag{2b}$$

By XOR'ing a known plaintext and ciphertext together per Equation 2a, the AES block key is obtained. By then XOR'ing this block key against an unknown ciphertext, for the length of the unknown ciphertext, discarding any remaining block key material, the unknown plaintext up was recovered, per Equation 2b.

This was confirmed across 2 Android devices, with different PINs and security parameters set on each, to prove that the key used is static, and not derived from the user's password. Therefore, the data is merely obfuscated. Anyone carrying out the above may decrypt any other user's Password Locker database trivially, using Equation 2b, since key is constant across all installations. This is of particular concern if users were to use the export functionality to store their passwords within cloud services, as any third party may determine their passwords using this technique.

E. Security Conclusions

This highlighted that while Video Locker does indeed use the AES cipher, it only encrypted the header of the MPEG-4 file, rather than its contents. It used static parameters for this encryption, for all files, irrespective of PIN used, leaking information between files. These static parameters were hard-coded into the application and therefore offer no security to users. Indeed, the use of CTR encryption with fixed initialisation vector also leaked other information, although the use of a static encryption key and IV meant that anyone with access to this widely-used software, or knowledge of how it operates, may decrypt files by any other user. More concerningly, Password Locker, another app from Handy Apps, uses similarly broken encryption to protect users passwords, while offering cloud backup facilities. The key for Password Locker is static across all devices, meaning that anyone may trivially decrypt anyone else's passwords and secret information. We demonstrate that this attack was possible, and note it did not take "trillions of years" to decrypt, as the developers asserted [14]. This is of particular concern for an application claiming to protect user passwords, especially given the off-device backup facility.

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

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

Fig. 7. Comparison of original file against Video Locker-protected file

VII. Video Locker (NewSoftwares.net)

Video Locker Advanced is an application by the developer NewSoftwares.net. To avoid ambiguity with the other application named Video Locker, investigated previously, we shall refer to this application as Video Locker Advanced, in-keeping with its package name. It had an average rating of 4.2/5 stars from 192 reviews, and had between 10,000 and 50,000 installs. Version 1.0.3 of the application was investigated, which was released in January 2016, and the latest available. The developers claimed to use "Encryption - The app locks your personal videos, prevents video hack.", and that it protects private videos "using fast encryption techniques" [15].

A. Operation of Application

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

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

Comparison of the original file with the protected file indicated that only the header of the video had been modified, with the first 100 bytes of the header flipped. Therefore, the fourth byte became the 96th byte, as shown in Figure 7 — the fourth byte `0x18` is seen at address `0x61`. The ASCII representation makes this reversal of the bytes clearer, as shown in the right column of Figure 7.

B. Security Conclusions

From the above, it was clear that Video Locker Advanced did not use the advanced encryption techniques which it claimed — this amounted to reversing the bytes of the file header. Concerningly, the application features Dropbox backup support [15], which may lead users to believe that they are uploading only encrypted data to Dropbox, when they are in fact uploading plaintext user files with merely minor obfuscation of the file headers.

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

Fig. 8. Schema of the GalleryVault database file table

VIII. Gallery Vault

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

The developers state that "The hidden file are all encrypted", and that "GalleryVault is a fantastic privacy protection app to easily hide and encrypt your photos, videos and any other files that you do not want others to see" [16].

A. Operation of Application

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

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

This database contained a tablet named `file`, which stored an unencrypted mapping between protected and unprotected files. The field `org_name` and `org_path` contained the original name and path of the file respectively, with the path also including the original filename. The database schema is shown in Figure 8.

A JPEG photograph was encrypted using GalleryVault. After extracting the encrypted file from the vault, it was compared to the original file. Figure 9 shows the results this comparison — only the first ten bytes of the file were found to differ, and had simply been set to have byte values of zero.

While recovering from this would be straightforward, only requiring identification of the correct header values, based upon the image dimensions, it was found that this was not

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

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

Fig. 9. Comparison of original file (upper) against GalleryVault-protected file (lower)

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

Fig. 10. GalleryVault database leaking original file header

necessary, on account of the leakage of the original header information within the GalleryVault database file.

Within the file table, the field `org_file_header_blob` contained the plaintext original file header, as shown in Figure 10. Therefore, with access to only the GalleryVault-protected file, and the backup database held in an adjacent directory, within the globally accessible shared storage, it was possible to immediately recover the file header, which can be compared against Figure 9 to be identical to the original file's header which was removed.

B. Security Conclusions

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

IX. Encrypt File Free

Encrypt File Free, by MobilDev, was the second application listed in the Play Store search for the query "encrypt". It had 50,000 to 100,000 installs, and an average rating of 3.5/5 from 278 reviews. Version 1.0.8 of the application, from November 2014, was the latest version available, and the version investigated.

A. Developer Claims

The developer of Encrypt File Free states "Encrypt File Free can encrypt and protect photos, videos, audios, pictures, doc, ppt, xls, pdf and other files using a password", and that "The encrypted file can only be opened with the correct password" [17]. They also state users should "Encrypt your files and not just hide them. This solution is better and safer than simply hiding files".

B. Operation of Application

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

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

Fig. 11. Plaintext selected for testing of Encrypt File Free

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

Fig. 12. Ciphertext output from Encrypt File Free for above test file

or weak and potentially breakable. The plaintext data is shown in Figure 11.

Upon encrypting this file, which contained a total of 112 bytes, a ciphertext of 1168 bytes was returned. Examination of this file highlighted that this file could likely be split into three chunks — an ASCII representation of a 16 bytes hex string, perhaps a hash like MD5, some unknown data, and finally data which appeared to be the encrypted content of the file. Figure 12 shows the resulting ciphertext output from Encrypt File Free, with some of the 1024-byte header truncated for readability.

The first 32 bytes were found to contain an ASCII representation of the plain, unsalted, MD5 hash of the user's PIN for the application. For the example shown in Figure 12, the PIN used was "111111" (as ASCII characters), and the MD5 hash shown in the first 32 bytes is an ASCII representation of $md5(111111)$. Therefore, the PIN was trivially exposed to anyone with access to a ciphertext produced by the application, on account of the ease of brute-forcing MD5 hashes. This may be a risk where users unintentionally expose their device or other PINs, as a result of the re-use of that PIN within this application. Also of interest was that if the original data and hash lengths $(112 + 32)$ were subtracted from the overall ciphertext length (1168), this indicated the middle section of the data occupied exactly 1024 bytes, perhaps suggesting padding or some form of fixed-length lookup table.

Indeed, by altering the PIN hash located at the header of the file, the same file could be decrypted by another device, which had never been in contact with the plaintext file, thus showing that the file was not being encrypted with the user PIN, and that its presence was merely for checking validity of the entered PIN. Therefore, the process was no better than storing the file in plaintext. In contrast to the other applications investigated however, Encrypt File Free did actually modify the body of the file, rather than merely the headers, although it does not offer any effective security.

C. Cryptanalysis of the Output

The output of the cipher was found to be very weak, and appeared to represent that of a monoalphabetic substitution cipher. Specifically, the pattern of blocks of data was visible in the output, with 16 differing values, then 16 values (say A), then 48 different values (say B), then a further 32 bytes of A. This pattern indicated that the structure of the input plaintext was remaining constant through to the ciphertext. This can be seen at the lower part of Figure 12, where the pattern of repeated bytes has been exposed from the plaintext through to the ciphertext.

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

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

By using a crib from a known plaintext and ciphertext mapping, and that a form of monoalphabetic substitution was taking place, it is possible to consider that the plaintext byte `0x00` from the first byte of the plaintext from Figure 11 was mapped to a ciphertext byte of `0x74`, per Figure 12. By observing that the byte `0x74` appears at offset `0x80` of this 256-byte header, it appears that the plaintext is obtained by subtracting the value `0x80`. This was verified for other byte values — the ciphertext byte `0x98` corresponded to plaintext `0x33`, and the ciphertext byte appeared at offset `0xb3`. Subtracting `0x80` from this resulted in the plaintext byte `0x33` as expected.

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

D. Security Conclusions

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

A written review from a user in September 2015 entitles "Unbreakable" states "Encryption still holds, after 2 months. (I hope) LoL" [17]. This does not appear to be the case, and the ciphertexts were not able to stand up to basic analysis, with a security level commensurate with that of a monoalphabetic substitution cipher. The application also leaked the unsalted, plain MD5 hash of the user's PIN in the header of each file, potentially exposing a user's PIN to other applications, which may be damaging if this were to be re-used in other scenarios, such as on a lock-screen or a bank card.

X. Discussion

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

The United States' FTC issues advice for app developers, encouraging them to consider security at the start of making an application [18], although much of this advice focuses more on privacy and data protection, rather than on proper implementation of cryptography. App developers have previously been found guilty of misleading practices, although these have typically focused on non-transparent fees to use applications, such as by sending premium rate SMS messages without making users aware [19].

It does appear to remain an open question, however, as to whether or not there is a legal case for claims of false or misleading advertising against mobile application developers, especially where an application is made available for free. While legislation exists to protect consumers from digital content sales [20], the rise in alternative business models, whereby the user does not pay directly for the application, but the developer receives money as a result of advertisments shown within the application to users, raises questions as to whether there is any recourse available for users against misleading claims made by developers.

XI. Conclusions

An analysis was conducted of a range of Android applications claiming to implement encryption to protect user

files from unauthorised access. This selection of applications included highly popular apps, and those by "Top Developers" on the Google Play Store. Every application here claimed to encrypt files for privacy or security, but most merely obfuscated files or removed their headers. Very few used actual encryption algorithms, despite their claims, and did those which did use encryption were using it with a static key, that was the same for every installation of the application. This was verified by repeating the experiment on a second device and ensuring "encrypted" files were able to be opened on both device. In that case, the use of an uninitialised counter-mode cipher with a static key, it was possible to recover all data encrypted by the application. Cryptanalysis of a non-standard algorithm for a monoalphabetic substitution cipher, based upon a mapping table held in plaintext in the header of the file, was carried out. This showed that basic analysis could be used to decode the data, without any requirement to understand the workings of the application.

These findings are of significant concern, as they show the security implications for users relying on software such as that which has been investigated here. Combined, the applications investigated here have potentially up to 117 million users, as reported by the rounded (and range-binned) figures within Google Play. These applications all claimed to use encryption, and a viable attack has been found to recover plaintext from each of them. In all cases, it was possible to recover the plaintext from the ciphertext without knowledge of any key or other security credential; these techniques serve merely as obfuscation, and do not offer the properties of properly implemented encryption, even where ciphers are named or indeed used, due to serious implementational flaws. Weaknesses such as these put sensitive or confidential data at risk of compromise, due to their false statements of security, and improper implementations. This highlights the need for open-source, audited security applications, and shows that users cannot necessarily trust app ratings, user counts, or developer claims, when considering security.

ACKNOWLEDGMENT

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

REFERENCES

[1] SANS Institute. (2014, June) Acceptable encryption policy. [Online]. Available: https://www.sans.org/security-resources/policies/general/pdf/acceptable-encryption-policy

[2] B. Schneier. (2011, April) Schneier's law. [Online]. Available: https://www.schneier.com/blog/archives/2011/04/schneiers_law.html

[3] B. Charles, "Passages from the life of a philosopher," 1864.

[4] A. Whitten and J. D. Tygar, "Why johnny can't encrypt: A usability evaluation of pgp 5.0." in *Usenix Security*, vol. 1999, 1999.

[5] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why eve and mallory love android: An analysis of android ssl (in) security," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 50–61.

[6] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, "The most dangerous code in the world: validating ssl certificates in non-browser software," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 38–49.

[7] G. Paul and J. Irvine, "Google's Android setup process security," in *Proceedings of Wireless World Research Forum Meeting 33*, September 2014.

[8] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in android applications," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 73–84.

[9] simple app 2016. (2016, January) Vlocker-hide videos. [Online]. Available: https://play.google.com/store/apps/details?id=com.simpleapp.vlocker

[10] SMSROBOT Ltd. (2015, October) Hide pictures & videos - fotox. [Online]. Available: https://play.google.com/store/apps/details?id=com.smsrobot.photox

[11] ——. (2015, October) Vault - hide photos/app lock. [Online]. Available: https://play.google.com/store/apps/details?id=com.smsrobot.vault

[12] Handy Apps. (2015, December) Hide photos in photo locker. [Online]. Available: https://play.google.com/store/apps/details?id=com.handyapps.photoLocker

[13] ——. (2015, December) Video locker - hide videos. [Online]. Available: https://play.google.com/store/apps/details?id=com.handyapps.videolocker

[14] ——. (2015, December) Password locker - encrypt data. [Online]. Available: https://play.google.com/store/apps/details?id=com.handyapps.passwordlocker

[15] NewSoftwares.net. (2016, January) Video locker - hide videos. [Online]. Available: https://play.google.com/store/apps/details?id=net.newsoftwares.videolockeradvanced

[16] ThinkYeah Mobile. (2015, December) Gallery vault-hide video&photo. [Online]. Available: https://play.google.com/store/apps/details?id=com.thinkyeah.galleryvault

[17] MobilDev. (2014, November) Encrypt file free. [Online]. Available: https://play.google.com/store/apps/details?id=com.acr.encryptfilefree

[18] FTC. (2013, February) Mobile app developers: Start with security. [Online]. Available: https://www.ftc.gov/tips-advice/business-center/guidance/mobile-app-developers-start-security

[19] C. Donnelly. (2012, September) Android app maker fined 50,000 for misleading consumers. [Online]. Available: http://www.itpro.co.uk/642616/android-app-maker-fined-50000-for-misleading-consumers

[20] Department for Busienss, Innovation & Skills. (2015, October) New rights for consumers when buying digital content. [Online]. Available: https://www.gov.uk/government/news/new-rights-for-consumers-when-buying-digital-content