ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

Advanced Mobiflage for PDE Enabled Storage

Neeraja M.Nair, Akhil S, Sheema M

Final Year M.Tech Student (Cyber Security), Dept. of Computer Science, SNGCE, Kerala, India

Final Year M.Tech Student (Cyber Security), Dept. of Computer Science, SNGCE, Kerala, India

Asst. Professor, Dept. of Computer Science, SNGCE, Kerala, India

ABSTRACT: Data confidentiality can be effectively preserved through encryption. In certain situations, this is inadequate, as users may be coerced into disclosing their decryption keys. Steganographic techniques and deniable encryption algorithms have been devised to hide the very existence of encrypted data. Here it examine the feasibility and efficacy of deniable encryption for mobile devices. To address obstacles that can compromise plausibly deniable encryption (PDE) in a mobile environment, we design a system called Advanced Mobiflage.Advanced Mobiflage enables PDE on devices by hiding encrypted volumes within random data in a devices free storage space.Here it allows users to provide a decoy password in a plausible manner.

KEYWORDS: PDE,Mobiflage, Confidentiality

I. INTRODUCTION

Smartphones and other computing devices are being widely adopted globally. With this increased use, the amount of personal/corporate data stored in mobile devices has also increased. Due to the sensitive of this data, all major mobile OS now include some level of storage encryption. Mobile phones have been extensively used to capture and publish many images and videos of recent popular revolutions and civil disobedience. Some vendors use file based encryption, such as Apple s iOS, while others implement a full disk encryption (FDE). Google introduced FDE in Android 3.0 (for tablets only); FDE is now available for all Android 4.x devices, including tablets and smartphones. While Android FDE is a step forward, it lacks deniable encryption a critical feature in some situations, e.g., when users want to provide a decoy key in a plausible manner, if they are coerced to give up decryption keys. Plausibly deniable encryption (PDE) was first explored by Canetti et al. [6] for parties communicating over a network. As it applies to storage encryption, PDE can be simplified as follows: different reasonable and innocuous plaintexts may be output from a given ciphertext, when decrypted under different decoy keys. The original plaintext can be recovered by decrypting with the true key. In the event that a ciphertext is intercepted, and the user is coerced into revealing the key, she may instead provide a decoy key to reveal a plausible and benign decoy message. The Rubberhose filesystem for Linux (developed by Assange et al. [3]) is the first known instance of a PDE-enabled storage system. Some real-world scenarios may mandate the use of PDEenabled storage e.g., a professional/citizen journalist, or human rights worker operating in a region of conflict or oppression. In a recent incident [45], an individual risked his life to smuggle his phone s micro SD card, containing evidence of atrocities, across international borders by stitching the card beneath his skin. Mobile phones have been extensively used to capture and publish many images and videos of recent popular revolutions and civil disobedience. When a repressive regime disables network connectivity in its jurisdiction, PDE-enabled storage on mobile devices can provide a viable alternative for data exfiltration.

With the ubiquity of smartphones, we postulate that PDE would be an attractive or even a necessary feature for mobile devices. Note, however, that PDE is only a technical measure to prevent a user from being punished if caught with contentious material; an adversary can always wipe/confiscate the device itself if such material is suspected to exist. Most OS lacks of deniable encryption a critical feature helps user in a forced situation to give their decryption keys. But they provide a decoy (fake) key which produce a different reasonable plaintexts output from a given cipher text, keeping main plaintext safe. PDE would be an attractive or even a necessary feature for all devices. Note, however, that PDE is only a technical measure to prevent a user from being punished if caught with contentious material; an adversary can always wipe/confiscate the device itself if such material is suspected to exist. Mobiflage s threat model and operational assumptions, and few legal aspects of using PDE in general .

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

The major concern with maintaining plausible deniability is whether the system will provide some indication of the existence of any hidden data. The threat model is mostly based on past work on desktop PDE solutions; we also include threats more specific to mobile devices. It provides storage encryption without deniability. The user will supply their decoy password at boot time to enter the standard mode. In this mode, the storage media is mounted in the default way (i.e., the same configuration as a device without Mobiflage). We use the terms decoy and outer interchangeably when referring to passwords, keys, and volumes in the standard mode. No solutions exist for any mainstream desktop OSes, although PDE support is apparently more important for these systems, as mobile devices are more widely used and portable than laptops or desktops. Also, porting mobile PDE solutions to desktop devices is not straight forward due to the tight coupling between hardware and software components, and intricacies of the system boot procedure. Here Advanced Mobiflage, a PDE-enabled storage encryption system for the desktop OS. It includes countermeasures for known attacks against mobile PDE . I explores and provide countermeasure to challenges more specific to using PDE systems in a mobile environment, including: collusion of cellphone carriers with an adversary; the use of flash-based storage as opposed to traditional magnetic disks; and file systems such as Ext4 (as used in Android) that are not so favorable to PDE. Advanced Mobiflage addresses several of these challenges. However, to effectively offer deniability, Advanced Mobiflage must be widely deployed, e.g., adopted in the mainstream Android OS. Main contributions include:

1. Explore sources of leakage inherent to mobile devices that may compromise deniable storage encryption. Several of these leakage vectors can be reduced by using existing desktop PDE solutions.
2. Advanced Mobiflage PDE scheme based on hidden encrypted volumes.
3. Provide a proof-of-concept implementation of Mobiflage . During the normal operation of Mobiflage (i.e., when the user is not using hidden volumes), there are no noticeable differences to compromise the existence of hidden volumes.
4. Address several challenges specific to Android.
5. We analyze the performance impact of our implementation during initialization and for data-intensive applications.

II. RELATED WORK

In this section, it discuss deniable encryption implementations related to Advanced Mobiflage, and provide an overview of available data encryption support as built into major desktop and mobile OSes. Several new ciphers, or enhancements to existing ciphers, have been proposed to create PDE schemes (e.g., [6, 35, 14, 29]). However, most of these proposals strive to enable PDE in network communications, and are not directly applicable to storage encryption. All major desktop OSes now offer storage encryption with FDE support (e.g., Windows BitLocker, Mac OS X FileVault, and Linux eCryptfs). FDE uses ciphers to encrypt entire storage devices or partitions thereof. Encryption is performed on small units, such as sectors or clusters, to allow random access to the disk. FDE subsystems typically exist at or below the file system layer and provide transparent functionality to the user. FDE schemes generally focus on providing strong confidentiality, making efficient use of the storage media (i.e., no excessive data expansion), and being relatively fast (i.e., no significant decrease in IO throughput). PDE adds another layer of secrecy over FDE.

Most mobile OSes also offer data encryption (but no PDE). BlackBerry devices use a password derived key to encrypt an internal storage AES key, and an ECC private key [39]. When a device is locked, the storage and ECC keys are wiped from RAM. Any messages received while the device is locked are encrypted with the ECC public key, and decrypted after unlock. Removable storage can also be encrypted. Per-file keys are generated and wrapped with a password derived key, and/or a key stored in the internal storage. iOS devices use a UID (device unique identifier) derived key to encrypt file system meta-data, effectively tying the encrypted storage to a particular device [2]. Per-file keys are stored in this meta-data and used to encrypt file contents. File keys can be wrapped with a UID derived key, or a UID and password derived key, depending on the situation (e.g., if the file must be opened while the device is locked, only a UID key is used). Unlike the transparency afforded by FDE, app developers must explicitly call the encryption API to protect app data [47]. The advantage of file based encryption over FDE is that the device is actually encrypted when the screen is locked (i.e., keys are wiped from RAM). Older Android 2.3 (Gingerbread) devices can make use of third party software (e.g., WhisperCore [51]) to encrypt the device storage. WhisperCore enhances the raw flash file system, YAFFS2, which has been superseded on current Android devices in favor of the Ext4 file system. Disk encryption software such as TrueCrypt [46] and FreeOTFE [17] use hidden volumes for plausible deniability. TrueCrypt offers encryption under several ciphers including AES, TwoFish, Serpent, and cascades of these ciphers in the XTS mode.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

On Windows systems, TrueCrypt can encrypt the OS system partition. A special boot loader is used to obtain the user s password and decrypt the disk before the OS is loaded. On Linux systems, similar functionality can be achieved using an early user-space RAM disk. This is not a straightforward solution for Android devices since the soft keyboard mechanism required to obtain the password is part of the OS framework and not immediately available on boot. A custom bootloader, implementing a soft keyboard, would be needed to capture the password (cf. [42]). The dm-crypt volume could then be mounted before loading the Android framework. We choose instead to work with the existing Android technique of partially loading the framework to access the built-in keyboard. TrueCrypt volumes contain a header at the very beginning of a volume. All fields in the header are either random data (e.g., salt) or are encrypted, giving the appearance of uniform random data for the entire volume. Unlike Android FDE, the cipher specification is not stored. Therefore, when a TrueCrypt volume is loaded, all supported ciphers and cascades of ciphers, are tried until a certain block in the header decrypts to the ASCII string TRUE. The header key is derived from the users passphrase using PBKDF2. If the header key successfully decrypts the ASCII string, then it is used to decrypt the master volume key, which is chosen at random during the volumes creation. A secondary header, adjacent to the primary header, is used when a hidden volume exists. The secondary header contains the same fields as the primary header, along with the offset to the hidden partition. When mounting a TrueCrypt volume, the hidden header is tested before the primary header. To combat leakage, when using hidden volumes, TrueCrypt recommends the use of a hidden OS. The hidden OS is currently only an option for the Windows implementation.

When encrypting a system volume for use with PDE, TrueCrypt creates a second partition and copies the currently installed OS to the hidden volume within. The user should only mount hidden volumes when booted into a hidden OS, to ensure that any OS/application-specific leakage stays within a deniable volume (e.g., logs, page file, hibernation file). When booted into a hidden OS, all unencrypted volumes and non-hidden encrypted volumes are mounted read-only. The alternative to a hidden OS for Linux, is to use a live CD when mounting hidden volumes. A hidden OS is not necessary in Advanced Mobiflage since the system volume on an Android device is mounted read-only, and we attach hidden volumes, or RAM disks, to all mutable volume mount-points to prevent leakage. There is a recent effort to port TrueCrypt to Android [8]. The current version (Dec. 2012) provides a command-line utility to create and mount TrueCrypt volume-container files (for rooted devices with LVM and FUSE kernel support). Hidden volumes are possible within these container files; but FDE/pre-boot authentication is not currently supported. Several leakage vectors also remain unaddressed (e.g., through file system structures, software logs, and network interfaces). Other Linux deniable implementations, such as RubberhoseFS [3], and Magikfs,7 employ techniques similar to StegFS for hiding data in file system free space; see Appendix A. Several of these projects are no longer maintained and existing implementations are also mostly incompatible with the modern Linux OS. The presence of specialized file system drivers designed to hide data would be a red flag to an adversary.

III. PROPOSED ALGORITHM

A. *Overview and Modes of Operation*

Advanced Mobiflage is implemented by hiding volumes in empty space .First fill the storage with random noise, to conceal the existence of additional encrypted volumes. Then create two adjacent volumes: a user data volume for applications and settings, and a larger auxiliary volume for accumulating documents, photos, etc. The exact location of the hidden volumes on the external storage is derived from the users deniable password.

Following modes of operation are there for Advanced Mobi- flage.

_ (a) Standard mode is used for day-to-day operation of the device. It provides storage encryption without deniability. The user will supply their decoy password at boot time to enter the standard mode. In this mode, the storage is mounted in the default way. It use the terms decoy and outer interchangeably when referring to passwords, keys, and volumes in the standard mode.

_ b) PDE mode is used only when the user needs to gather/store data, the existence of which may need to be denied when coerced. The user will supply their true password during system boot to activate the PDE mode. It use the terms true, hidden and deniable interchangeably when referring to passwords, keys, and volumes in the PDE mode.

B. *Storage Layout*

The entire file is encrypted with a decoy key and formatted for regular use; we call this the outer volume. Then additional file systems are created at different offsets within the disk and encrypted with different keys; these are

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

referred to as hidden volumes. Some hidden volumes may be decoys, but at least one hidden volume will contain the actual sensitive data and be encrypted with the true key.

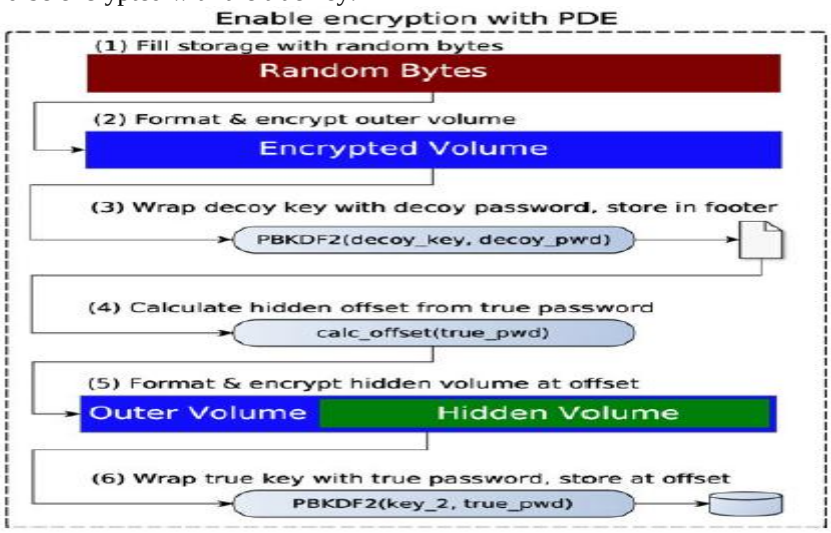

Figure 1: Enabling Encryption with PDE

Since the outer volume is filled with random noise before formatting, there are no distinguishing characteristics between empty outer-volume blocks and hidden volume blocks. When the outer volume (or a hidden decoy volume) is mounted, it does not reveal the presence or location of any other hidden volumes. All hidden volumes are camouflaged amongst the random noise. The disk can be thought of as the concatenation of encrypted volumes, each with a different key: EK1(V ol1)||EK2(V ol2)||...||EKn(V oln)

Here, EK() represents a symmetric encryption function with key K and || represents concatenation. When the file is decrypted with a given key, the other volumes will appear to be uniformly random data. When the user is coerced, she can provide the outer volume key and claim that no other volumes exist:
DK1(V ol1||V ol2||...||V oln) = V ol1||Rand

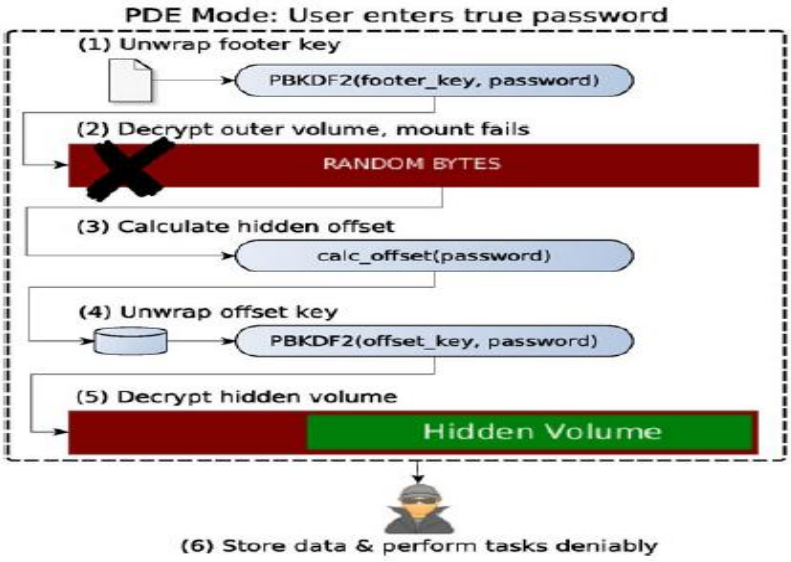

Figure 2: PDE Mode

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

Here, DK( ) represents the symmetric decryption function with key K (corresponding to EK()) and Rand represents data that cannot be distinguished from random bits. Therefore, a forensic analysis of the decrypted outer volume will not indicate the existence of hidden volumes. At this time, the user can provide decoy keys for other hidden volumes and insist that all the volumes have been exposed. Revealing the existence of any hidden volume may either help or hinder the user, depending on the situation.

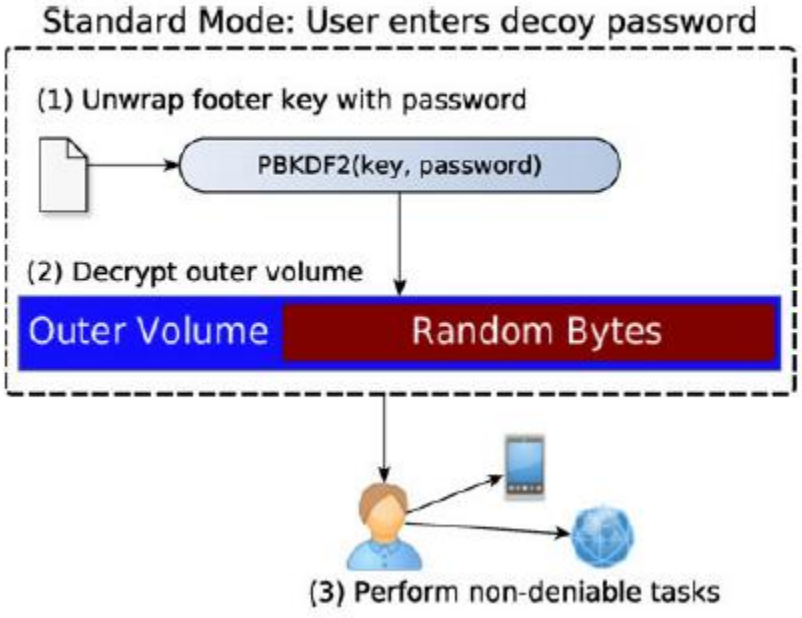

Figure 3: Standard Mode

C. *Offset Calculation*

The offset to a hidden volume is generated as follows:

offset = 0.75 * vlen(H(pwd||salt) mod (0.25 * vlen))

Here, H is a PBKDF2 iterated hash function [26], vlen is the number of 512-byte sectors on the logical block storage device, pwd is the true password, and salt is a random salt value for PBKDF2. The salt value used here is the same as for the outer volume key derivation (i.e., stored in the encryption footer). Thus, avoid introducing an additional field in the default encryption footer that may indicate the presence of hidden volumes. The generated offset is greater than one half and less than three quarters of the disk; i.e., the hidden volume s size is between 25-50 percent of the total disk size (assuming only one hidden volume is used). This offset is choosen as a balance between the hidden and outer sizes: the outer volume will be used more often, the hidden volume is used only when necessary. To avoid overwriting hidden files while the outer volume is mounted, it is recommended the user never fills their outer volume beyond 50 percent. Deriving the offset in the above manner allows us to avoid storing it anywhere on the disk, which is important for deniability. Introducing a new field to store the offset would reveal the use of Advanced Mobiflage PDE, so choose to derive the offset from the password instead.

D. *User Steps*

Here, describe how users may interact with Advanced Mobiflage, including initialization and use. Users must first enable device encryption with PDE (e.g., through settings GUI). Assuming a single hidden volume is used, the user then enters the decoy and true passwords, for the outer and hidden volumes respectively. Advanced Mobiflage then creates the hidden volumes, performs in-place encryption of the internal storage (if chosen) and reboots when complete.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

Unlike Android FDE, Advanced Mobiflage must initialize the external storage with random data for deniability. However, the initialization step will likely be performed only occasionally. For normal day-to-day use (e.g., phone calls, web browsing), the user enters the decoy password during authentication to activate the standard mode. All data saved to the device in this mode will be encrypted but not hidden. It is important for the user to regularly use the device in this mode, to create a digital paper trail and usage timeline which may come under scrutiny during an investigation. The user gains plausibility by showing that it is frequently used in this mode; i.e., she can demonstrate apparent compliance with the adversary s orders. When the user requires the added protection of deniable storage, they will reboot it and provide their deniable password when prompted. In the PDE mode, they can transfer documents from another device, or take photos and videos. Note that app/system logs in this mode are hidden or discarded; however, there is still a possibility of leakage through network interfaces.

After storing or transferring files to the deniable storage, the user should immediately reboot into the standard mode. The files are hidden as long as the device is either off, or booted in the standard mode. If the user is apprehended with the device in the PDE mode, deniability is lost. Depending on the situation, the user can provide additional decoy passwords, when faced with continued coercion. A rational adversary may not punish the user if they have no reason to believe that (further) hidden data exists on the device. Assuming the user can overcome any coercion the adversary attempts, and does not reveal the true key, the adversary will have no evidence of the hidden data.

E. *Security Analysis*

In this section, we evaluate Advanced Mobiflage against known attacks and weaknesses.

a) Password guessing. We rely on the user to choose strong passwords to protect their encryption keys. The current Android encryption pre-boot authentication times-out for 30 seconds after ten failed password attempts. The timeout will slow an online guessing attack, but it may still be feasible, especially when weak passwords are used. An offline dictionary attack is also possible on an image of the device s storage. The adversary does not know the password to derive the offset, but the salt is found in the Android encryption footer. The salt is used with PBKDF2, and is a precaution against pre-generated dictionaries and rainbow tables. The salt cannot be stored at the hidden offset as it is used in the offset calculation. Using the same salt value for both modes enables the adversary to compute one dictionary of candidate keys (after learning the salt), to crack passwords for both modes.

b) Cipher issues. An implementation flaw can expose FDE ciphers to a theoretical watermarking attack that has been documented for software such as LUKS [9]. The issue occurs when the disk is sufficiently large and the size of the disk sector index (n) is small. For example, if n is a 32-bit integer, and there are more than 2 32 512-byte sectors on the disk, the value of n will eventually roll-over and repeat itself. If the adversary can create a special file with duplicate plaintext blocks at correct locations and convince the user to store the file in their hidden volume, then the adversary can demonstrate the existence of a hidden volume. In the given example, the duplicate plaintext blocks would need to be repeated at 2TB intervals. The adversary will not know what the corresponding ciphertext blocks will be, but finding identical ciphertexts spaced at the correct distance would be strong evidence. This is an implementation issue, and not an issue with the cipher algorithm itself. The problem occurs for all FDE ciphers, including XTS and CBCESSIV, that use a sector index smaller than the total number of disk sectors. To mitigate this problem, a longer integer (e.g., 64-bit) is commonly used for the sector index. We use the 64-bit sector index available in dm-crypt which will not roll over until 8192 Exabytes.

IV. RESULTS AND DISCUSSION

With Advanced Mobiflage, the implementation challenges of PDE enabled devices, which may be more useful to regular users and human rights activists are identified. Advanced Mobiflage s design is partly based on the lessons learned from known attacks and weaknesses of mobile PDE solutions. To address some of these challenges, it need the user to comply with certain requirements. Here it compiled a list of rules the user must follow to prevent leakage of information that may weaken deniability. Hence Advanced Mobiflage encourage further investigation of PDE-enabled systems.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

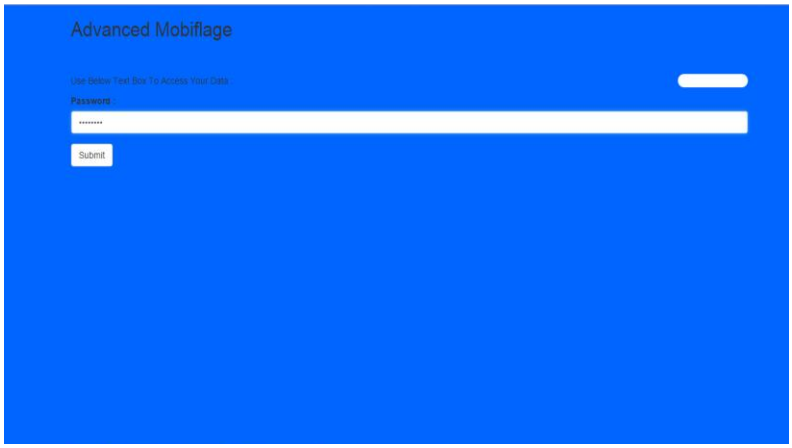

Figure 4: Enabling User Mode

Figure 4 shows the user interface to enter the password. The user on standard mode enters the decoy password and retrieves the standard data and files. The user on PDE mode enters the true password and retrieves the hidden data and files. The retrieved data is shown in figure 5.

Figure 5: User Mode Enabled

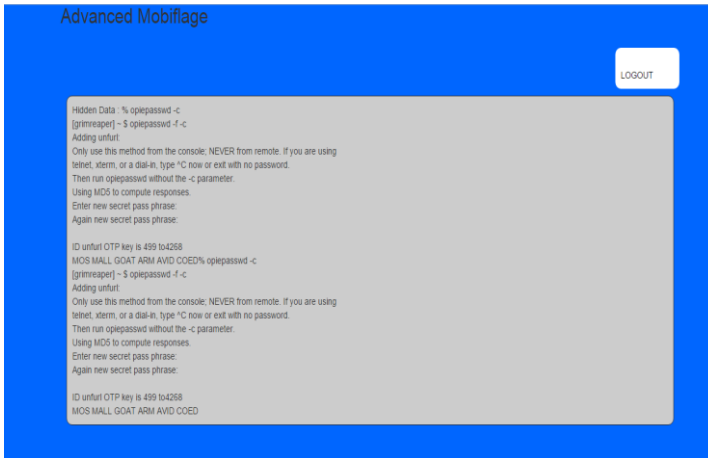

Figure 6: PDE Mode Enabled

V. CONCLUSION AND FUTURE WORK

Most devices are increasingly being used for capturing and spreading images of popular uprisings and civil disobedience. To keep such records hidden from authorities, deniable storage encryption may offer a viable technical solution. Such PDE-enabled storage systems exist for whole device encryption of operating systems. With Advanced Mobiflage, explore design and implementation challenges of PDE for mobile devices, which may be more useful to regular users and human rights activists. Advanced Mobiflage s design is partly based on the lessons learned from known attacks and weaknesses of mobile PDE solutions. It also consider unique challenges in the mobile environment (such as ISP or wireless carrier collusion with the adversary). To address some of these challenges, it need the user to comply with certain requirements. Here it compiled a list of rules the user must follow to prevent leakage of

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

information that may weaken deniability. Hence Advanced Mobiflage encourage further investigation of PDE-enabled systems.

REFERENCES

1 A. Skillen and M. Mannan, On Implementing Deniable Storage Encryption for Mobile Devices, Proc. Network and Distributed System Security Symp. (NDSS 13), Feb. 2013.
2 comScore, comScore Reports September 2012 U.S. Mobile subscriber Market Share, 2012..
3 R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky, Deniable Encryption, Proc. 17th Ann. Int l Cryptology Conf. Advances in Cryptology (CRYPTO 97), 1997.
4 J. Assange, R.-P. Weinmann, and S. Dreyfus, Rubberhose: Cryptographically Deniable Transparent Disk Encryption System, Project Website: http://marutukku.org/, 1997.
5 Toronto Star, How a Syrian Refugee Risked His Life to Bear Witness to Atrocities, News Article, http://www.thestar.com/ news/world/article/1145824, Mar. 2012.
6 TrueCrypt, Free Open Source on-the-Fly Disk Encryption Software, version 7.1a, http://www.truecrypt.org/, July 2012.
7 A. Czeskis, D.J.S. Hilaire, K. Koscher, S.D. Gribble, T. Kohno, and B. Schneier, Defeating Encrypted and Deniable File Systems: TrueCrypt v5.1a and the Case of the Tattling OS and Applications, Proc. USENIX Workshop Hot Topics in Security (HotSec 08), 2008.
8 C. Fruhwirth, New Methods in Hard Disk Encryption, technical report, Vienna Univ. of Technology, http://clemens.endorphin. org/nmihde/nmihdeA4-ds.pdf, July 2005.
9 R.M. Needham and M.D. Schroeder, Using Encryption for Authentication in Large Networks of Computers, Comm. ACM, vol. 21, no. 12, pp. 993-999, 1978.
10 D. Dolev and A.C. Yao, On the Security of Public Key Protocols, IEEE Trans. Information Theory, vol. IT-29, no. 2, pp. 198-208, Mar. 1983. 11 R.-P. Weinmann, Baseband Attacks: Remote Exploitation of Memory Corruptions in Cellular Protocol Stacks, Proc. USENIX Workshop Offensive Technologies (WOOT 12), 2012.
12 TheRegister.co.uk, UK Jails Schizophrenic for Refusal to Decrypt Files, News Article, http://www.theregister.co.uk/2009/11/24/ ripajfl/, Nov2009.