

# FROST

## Forensic Recovery of Scrambled Telephones

Tilo Müller, Michael Spreitzenbarth, and Felix C. Freiling

Department of Computer Science  
Friedrich-Alexander University of Erlangen-Nuremberg

October 2012

**Abstract.** At the end of 2011, Google released version 4.0 of its Android operating system for smartphones. For the first time, Android smartphone owners were supplied with a disk encryption feature that transparently scrambles user partitions, thus protecting sensitive user information against targeted attacks that bypass screen locks. On the downside, scrambled telephones are a nightmare for IT forensics and law enforcement, because once the power of a scrambled device is cut any chance other than brute force is lost to recover data.

In this paper we present FROST, a tool set that supports the *forensic recovery of scrambled telephones*. To this end we perform cold boot attacks against Android smartphones and retrieve disk encryption keys from RAM. We show that cold boot attacks against Android phones are generally possible for the first time, and we perform our attacks practically against Galaxy Nexus devices from Samsung. To break disk encryption, the bootloader must be unlocked before the attack because scrambled user partitions are wiped during unlocking. However, we show that cold boot attacks are more generic and allow to retrieve sensitive information, such as contact lists, visited web sites, and photos, directly from RAM, even though the bootloader is locked.

## 1 Introduction

In 2011, 83 percent of the American adults had a cell phone from which 42 percent had a phone that can be classified as smartphone [35]. Android is today the most common smartphone platform, followed by iOS, Blackberry, and Windows. Regardless of the platform in use, smartphones are an important technology to keep employees connected to their company. Most consumers use their smartphones for both business and personal use, and a missing device has severe consequences for both private users and their employers.

### 1.1 Physical Smartphone Security

In 2011, AVG and the Ponemon Institute presented findings of their *Smartphone Security Survey* [30] considering the smartphone usage of 734 consumers. 84 percent of the surveyed consumers use their smartphone for both business and personal purposes. For example, 89 percent use their smartphone for email and

82 percent use it also for business email. 66 percent admit they keep a significant amount of personal data on their phone, 38 percent use it for electronic payments, and 14 percent use it even for online banking. Sensitive information like those are at risk when a phone is *physically* lost or stolen. Although malicious apps can be used to spy upon user data remotely [1, 11], we focus on the *physical security layer* of smartphones.

In another study from 2011, McAfee and the Ponemon Institute presented results of their survey *The Lost Smartphone Problem* [31] on 439 U.S. organizations from both industry and public sectors. The study objectively determined that in a 12-month period 142,708 out of 3,297,569 employee smartphones were lost or stolen, i.e., 4.3 percent per year. 5,034 of these smartphones are known to be theft, while the others are just “missing”. Only 9,298 smartphones could be recovered within the time of the study. 47 percent of the smartphones were lost at home or in hotel rooms, 29 percent were lost while traveling, and 13 percent were lost in the office.

Personal and corporate data are not only at risk when smartphones become intentionally stolen by criminals, but also if they are lost and found by ordinary people. In 2012, researchers from Symantec presented their results of the *Smartphone Honey Stick Project* [40]. In this project, 50 smartphones were intentionally lost in cities around the U.S. and in Canada. The phones were prepared with a contact list, an app called “online banking”, files named “salaries” and “saved passwords”, an email account, personal photos, and more. Additionally, the phones were loaded with logging software so that Symantec could keep track of all activities. The study came to the result that 96 percent of all smartphones were accessed by their finders. 89 percent of the finders accessed personal data, and 83 percent accessed corporate data. For example, 43 percent clicked on the online banking app, 53 percent clicked on the salaries file, 60 percent checked personal emails, and 72 percent checked personal photos.

## 1.2 Protection Mechanisms against Physical Access

Although lost and stolen smartphones are widely known to be a security concern, risky behaviors and weak security postures are frequently in place today [32]. Using screen locking mechanisms can already prevent ordinary people from accessing sensitive data, but less than 50 percent of consumers use PIN-locks or passwords today [30]. To the contrary, 57 percent of all smartphones are not protected with any security mechanism [31]. The most frequently stated reason for that is the inconvenience arising from screen locks.

Even worse, also if PIN-locks or passwords are used, only opportunistic attacks can be defeated. Given physical access, we distinguish two types of attacks against smartphones: opportunistic and targeted attacks. In *opportunistic* attacks an adversary finds or steals a smartphone and immediately tries to retrieve personal (or corporate) data from it. This scenario is withstood by common screen locking mechanisms like PIN-locks, passwords, and pattern-locks, or by new technologies like finger gestures, face recognition, and speech recognition [24]. However, more careful adversaries, such as forensic experts from law enforcement, can carry out

*targeted* attacks. Targeted attacks do not stop at the barrier of screen locks, but try to retrieve data through more advanced methods. For example, although screen locks are in use, data can be retrieved from flash memories without using telephone software. Considering *external* flash memory (i.e., SD cards) this task becomes trivial, but considering *internal* flash memory, the attack requires expert knowledge in electronics or computer science, e.g., about the JTAG interface and tools like the *RIFF Box* [33].

Targeted attacks can be counteracted through anti-theft solutions with *remote wipe* functionality to delete sensitive contents of lost and stolen smartphones [3]. Such anti-theft solutions are available from all notable anti virus vendors, like Kaspersky, F-Secure, and Symantec. However, to reduce the risk of sensitive data being accessible, encryption technologies based on AES [14] are more reliable. Besides third party apps like the *Encryption Manager*, which works on file basis, Android supports *full disk encryption* (FDE) to scramble entire user partitions transparently. The encryption feature is available since Android 4.0 which was released in October 2011. Apple’s iOS has built-in support for data encryption even longer and enables it automatically if a password is set [5].

With a disk encryption solution in use, targeted attacks on smartphones should fail because only scrambled data can be retrieved when bypassing the screen lock. Hence, encryption technologies are an important feature for personal smartphones, and an essential requirement for future business devices. However, they are a nightmare for IT forensics. It seems as law enforcement eventually lost the crypto wars because there are no restrictions against strong cryptography on the mass-market today. For Apple and BlackBerry phones, it is already rumored that “the rise of AES hardware encryption in devices such as the iPhone and BlackBerry has made it all but impossible for the government forensic experts to extract desired info” [13].

### 1.3 Contributions: Breaking Android FDE with Cold Boot Attacks

In the work at hand we investigate the security of Android’s encryption feature. Android is the most widespread mobile platform in use today, and it is most amenable for an in-depth security analysis because it is open source and Linux-based, such that kernel-hacking (as required by our implementation) becomes possible. In short, for the case of Android we show that the encryption solution is vulnerable to *cold boot attacks* [18]. In this sense, Android’s encryption solution does withstand opportunistic attacks, but it does not withstand targeted attacks. More detailed, our contributions are:

- We prove that cold boot attacks against Android/ARM-based devices like tablets and smartphones are possible in general.
- In particular, we present the tool FROST (*Forensic Recovery of Scrambled Telephones*) for Galaxy Nexus devices. FROST is a recovery image that is to be installed into the recovery partition of a smartphone *after* we got physical access to it. It is not necessary to have FROST installed before the attack.

- On devices with an unlocked bootloader, FROST can recover encryption keys from RAM and decrypt the user partition directly on the phone. Similarly, we integrated a bruteforce option against short PINs that runs directly on the phone and decrypts the user partition, too.
- On devices with a locked bootloader, retrieving encryption keys from RAM is pointless since the unlocking process wipes all user data. However, FROST can also be deployed to take complete memory dumps of the phone in order to analyze them offline. We were able to recover recent emails, photos, and visited web sites from physical RAM dumps.
- We evaluate how the operating temperature of a Galaxy Nexus device correlates with the success of cold boot attacks.

A tutorial, a photo series, source codes (published under GPL v2 [36]), and compiled binaries are available at [www1.cs.fau.de/frost/](http://www1.cs.fau.de/frost/).

## 1.4 Paper Outline

The remainder of this paper is structured as follows: In Sect. 2 we provide background information about Android’s encryption solution and about cold boot attacks. In Sect. 3 we describe technical details behind the implementation of our recovery image FROST, and in Sect. 4 we evaluate the effectiveness of cold boot attacks against scrambled telephones. We conclude with an outlook over possible countermeasures and future works in Sect. 5.

# 2 Background Information

We now provide necessary background information about the support of full disk encryption in Android 4.0 and subsequent versions (Sect. 2.1), as well as the cold boot attack on encryption keys (Sect. 2.2). We also give a brief summary about technical details of our device under test, the Samsung Galaxy Nexus (Sect. 2.3).

## 2.1 Disk Encryption in Android 4.0

Google’s *Android operating system* is a Linux-based open source project that can be adapted by smartphone and tablet manufacturers and can freely be shipped together with new devices. The initial beta version of Android was released in November 2007, and the first stable version 1.0 followed in September 2008. Versions appear regularly and in October 2011, Android 4.0 alias Ice Cream Sandwich, in short Android ICS, was published. Since Android 4.0, Google maintains an ARM-specific fork of the Linux kernel version 3.0.

With Android 4.0, support for full disk encryption was introduced. While third party apps that extend the functionality of Android-based smartphones are primarily written in Java, disk encryption resides entirely in system space and is written in C. It is based on the known FDE solution *dm-crypt* [34] which is available in Linux mainline kernels since years. Dm-crypt relies on the *device-mapper* infrastructure and *Crypto API* and thus, it provides a flexible way to

encrypt block devices transparently. It creates a virtual encryption layer on top of all kinds of abstract block devices, including real devices, logical partitions, loop devices (files), and swap partitions. Writing to a mapped device gets encrypted and reading from it gets decrypted. LUKS [15], for example, makes use of the dm-crypt backend, too. Although dm-crypt is suitable for whole disk encryption, Android does *not* encrypt whole disks but user partitions mounted at `/data` only.

Dm-crypt is kept modular and it supports ciphers and modes of operation known from the Crypto API, including AES, Twofish and Serpent, as well as CBC, XTS, and ESSIV. Android 4.0 makes use of the dm-crypt mode `aes-cbc-essiv:sha256` with 128-bit keys [4]. The AES-128 *data encryption key* (DEK, also *master key*) is encrypted with an AES-128 *key encryption key* (KEK) which is in turn derived from the user PIN or password through the *password-based key derivation function 2* (PBKDF2) [39]. Using two different keys, namely the DEK and the KEK, renders cumbersome re-encryption in the case of PIN or password changes unnecessary. The encrypted DEK as well as an *initialization vector* (IV, also *salt*) for PBKDF2 are random 128-bit numbers taken from `/dev/urandom` and get stored inside the crypto footer. The *crypto footer* of a disk can either be an own partition or it can be placed at the very last 16 kBytes of an encrypted partition.

Unlike iOS, which does automatically activate disk encryption when a passcode is set, Android’s encryption is disabled by default. Activating it manually, takes up to an hour for the initial process to encrypt the disk and cannot be undone. Furthermore, it can only be activated if PIN-locks or passwords are in use. PINs consist of 4 to 16 numeric characters, and passwords consists of 4 to 16 alphanumeric characters with at least one letter. 4-digit PINs are still the most widespread screenlock today. New screen locking mechanisms like pattern-locks and face recognition are less secure, and thus Google forbids them in combination with FDE. Pattern-locks, for example, can be broken by so called *Smudge Attacks* [6], and face recognition can simply be tricked by showing a photo of the smartphone owner [23].

## 2.2 The Cold Boot Attack

On PCs, adversaries with physical access can perform one of several *physical access attacks* against FDE. Besides *evil maid attacks* [20] (also known as *bootkits*) and *DMA attacks* over FireWire [7], PCIe [12], or Thunderbolt [8], *cold boot attacks* constitute such a threat. Cold boot attacks are known since 2008, when Halderman et al. [18] showed how to exploit the *remanence effect* of DRAM [17] in order to recover encryption keys. The remanence effect says that RAM contents fade away gradually over time, as slower as colder the RAM chips are. Due to this effect, keys can be restored from main memory through rebooting a PC with malicious USB drives, or by replugging RAM chips physically into another PC. Such attacks are generic and constitute a threat to *all* software-based FDE technologies, including dm-crypt for Linux [34], BitLocker for Windows [25], and the cross-platform utility TrueCrypt [38]. A recent study from 2011 [10] confirms the practicability of cold boot attacks against x86 PCs.

However, it has not been reported yet if cold boot attacks are applicable against ARM-based devices such as smartphones and tablets, or against Android devices in particular. We conjecture such devices *are* vulnerable, because Android’s underlying encryption solution dm-crypt is already known to be vulnerable. Technically, it makes no difference if dm-crypt is running on ARM or an x86 architecture, because the vulnerability relies in the AES key schedule that is stored inside RAM. AES key schedules can be identified by recovery tools like *aeskeyfind* [19] that search for suspicious patterns of a schedule in RAM. Dm-crypt is vulnerable to such tools, because it creates the AES key schedule initially inside RAM during boot and it gets lost only if power is cut.

The open question we had to answer was, how to obtain a physical RAM dump from Android devices if the screen is locked? Unlike x86 PCs, we can neither plugin a bootable USB drive, nor can we simply unplug RAM chips physically in order to read them out in a second device.

### 2.3 Samsung Galaxy Nexus

We have chosen the Galaxy Nexus from Samsung as device under test because it is the first device that was available with Android 4.0 and consequently, it is the first Android-based smartphone with FDE support. Furthermore it is an official Google phone, meaning that it comes with a stock Android from Google that is not modified by its hardware vendor. Official Google phones are most amenable for an in-depth security analysis, and flaws can be generalized best to a wider class of devices. For the Samsung Galaxy SII, for example, we were able to recover AES keys from RAM, but we could not instantly decrypt the user partition because Samsung’s encryption mode seems to differ from that of official Android releases. For Sony’s Xperia mini, on the other hand, we were not even able to turn on encryption, although Android 4.0 was installed, presumably because Sony disabled it for performance reasons.

More specifically, we own a GSM/HSPA variant of the Galaxy Nexus (GT-9250), codename *tuna/maguro*. There are slightly different CDMA/LTE models available (codename *tuna/toro*) but we conjecture that our implementations run on both variants. The Galaxy Nexus family comes with an OMAP4 chip from Texas Instruments (4460) such that it has a Cortex-A9 CPU implementing the ARMv7 instruction set [37].

The partition layout of a scrambled Galaxy Nexus is given in Fig. 2.3. Most of the thirteen partitions can be ignored for our purpose, except userdata, metadata and recovery. The userdata partition contains the encrypted filesystem we want to break. The metadata partition is the crypto footer holding necessary information about the encryption, as stated in Sect. 2.1. And the recovery partition is a partition that holds a *second* bootable Linux system besides the system partition with Android. This partition can be compared best with a rescue system from ordinary PCs, allowing basic operations on the hard disk without booting into full Android.

The recovery partition plays a role in our cold boot attack because we make use of it to boot our own, malicious recovery image. As we are not able to boot

block device	partition name	description
/dev/block/mmcblk0p1	xloader	bootloader code
/dev/block/mmcblk0p2	sbl	bootloader code
/dev/block/mmcblk0p3	efs	static information like IMEI
/dev/block/mmcblk0p4	param	boot parameters
/dev/block/mmcblk0p5	misc	system settings like carrier ID
/dev/block/mmcblk0p6	dgs	unknown ( <i>zero filled on all devices</i> )
/dev/block/mmcblk0p7	boot	boot code
/dev/block/mmcblk0p8	recovery	recovery image
/dev/block/mmcblk0p9	radio	radio firmware (GSM)
/dev/block/mmcblk0p10	system	Android operating system
/dev/block/mmcblk0p11	cache	cache (e.g., for user apps)
/dev/block/mmcblk0p12	userdata	user data (encrypted)
/dev/block/mmcblk0p13	metadata	crypto footer

**Fig. 1:** Partition layout of scrambled Samsung Galaxy Nexus devices.

smartphones from external USB devices, as we could do on PCs, we had to find another way to execute system code. In short, we build a recovery image that walks through all physical memory pages in order to find AES keys. We flash this image to the Galaxy Nexus device and then reboot; this can be done *after* we found the device. However, if the bootloader is locked it must first get unlocked in order to flash. Unfortunately, the unlocking process wipes the userdata and cache partition and thus, searching for the AES key after unlocking becomes pointless (although still possible). We verified that the Galaxy Nexus actually *wipes* the userdata and cache partition, meaning that it zero-fills them.

The wiping process implemented by Google is commendable as it even renders data recovery in the case of non-encrypted partitions difficult. However, the first series of Galaxy Nexus devices, which could be ordered via Google’s Play Store, had an implementation flaw: they did *not* wipe the user partition after unlocking the bootloader [42]. Furthermore, we conjecture that there is an above-average number of Galaxy Nexus devices with unlocked bootloaders, because it is a developer phone. Other devices, like the Samsung Galaxy SII, even come with unlocked bootloaders out-of-the-box [41]. For all Android-based smartphones we find with an unlocked bootloader, or that we can unlock without wiping the user partition, we can perform cold boot attacks against disk encryption. For all other devices, we can still perform cold boot attacks to retrieve information from RAM such as contact lists, photos, and visited websites.

### 3 Implementation of the FROST Recovery Image

We now present the implementation of our utility FROST (*Forensic Recovery Of Scrambled Telephones*). Technically, FROST is a set of tools that are bracketed together into an easy-to-use recovery image for Galaxy Nexus smartphones. The recovery image displays a graphical user interface that allows IT practitioners,

for example, from law enforcement, to recover encryption keys and to unlock the scrambled user partition with only a few clicks. Furthermore, we implemented an option for bruteforce attacks against weak PINs, and we implemented an option to save full RAM dumps to PCs in order to analyze them offline.

### 3.1 The Linux Kernel Module

The heart of our FROST project is its loadable Linux kernel module (LKM) of the same name. The FROST LKM is based on the known userland utility *aeskeyfind* [18, 19] that searches for AES keys in a given memory image of x86 PCs. Contrary to that, FROST is implemented for ARM and searches for AES keys *on-the-fly*, i.e., directly on the phone. Without the FROST LKM, the process of cold boot attacks would be quite cumbersome, because (1) a memory image of the phone would have to be taken, e.g., with the *Linux Memory Extractor* (LiME) [21], (2) this memory image would have to be transferred to a PC via USB, and (3) the image would have to be analyzed with *aeskeyfind*. The entire process would take up to ten minutes. FROST, on the other hand, recovers AES keys in about 10 seconds. An exemplary FROST output is given in Fig. 2.

The FROST module was developed and tested on Galaxy Nexus devices, but it works on other Android-based smartphones, too. It is a part of our project which is *platform-independent* meaning that it even runs on non-Android systems. For example, we have successfully tested our module on a PandaBoard with Ubuntu. In general, FROST can be used on all ARM devices where you have a Linux shell with root access. It is licensed under GPL v2 [36] and is together with other information available on our webpage. You may use it independent of the FROST recovery image.

The FROST LKM basically supports two search modes: *quick search* and *full search*. Quick search is highly optimized for Galaxy Nexus devices and looks for AES keys at certain RAM addresses. In detail, we have chosen the physical address space `0xc5000000` to `0xd0000000` because all our tests revealed that AES keys are placed in this range. In quick search mode, the recovery process finishes within seconds. This mode, however, might fail on other devices because the search space might be too specific. Therefore, we implemented a full search mode considering the entire physical RAM. Additionally, full search mode uses a sliding window mechanism that looks at each physical RAM page twice. In quick search mode, AES key schedules which are spread over multiple physical pages might be missed.<sup>1</sup> As a downside, the full search mode runs up to ten minutes. It is generally a good idea to use quick search first and full search only if that fails.

### 3.2 PIN Cracking through Bruteforce

PINs are still the most frequent screen lock in use today. But long PINs are too inconvenient for most people that work on their phones on a daily basis, because

---

<sup>1</sup> In practice, however, this is very unlikely. We never observed any AES key schedule that was spread over two pages. (The page size in Android is 4096 bytes.)



```
adb> insmod frost.ko fullsearch=0 ; dmesg

key-32: 4ee35476397b76905828a89f3d9b872f
       : ccb6671af6eebffe94ea1bc87c0948e4
key-32: 4ee35476397b76905828a89f3d9b872f
       : ccb6671af6eebffe94ea1bc87c0948e4
key-16: bcdbc55cf809cb5989e58a40ecbb7164
key-16: bcdbc55cf809cb5989e58a40ecbb7164

Summarizing 4 keys found.
```

**Fig. 2:** AES keys recovered by the FROST LKM.

```
adb> ./crackpin

magic: DOB5B1C4
encdek: 3c4ac402c6095ed46cf4f1e2281a1f3e
salt: 19043211840adfdde95110c7f99263d6c

>> KEK: 2165534cc66099714a8226753d70b576
>> IV: 05cb47cf3a98d77e563bb4cfcde791aa
>> DEK: bcdbc55cf809cb5989e58a40ecbb7164
>> PIN: [2323]
```

**Fig. 3:** Key and PIN recovered by brutefore.

they must be entered for each interaction with the device, e.g., for giving a call, for writing a message, and for taking a photo. Consequently, people commonly use short PINs of only 4 digits. That is concerning, because in Android the screen lock PIN necessarily equals the PIN that is used to derive the disk encryption key. Hence, short PINs are just another weak point of Android’s encryption feature besides cold boot attacks.

In 2012, Cannon et al. [9] presented details about Android’s encryption system and gave instructions on how to break it with bruteforce attacks against the PIN. They published their findings in form of a Python script that breaks Android encryption offline, meaning that it runs on an x86 PC after the userdata and metadata (crypto footer) partition have been retrieved “somehow”. Basically, we re-implemented the Python script in C and cross-compiled it for the ARM architecture, so that we can perform efficient attacks directly on the phone without the need to download the user partition first. To this end, we cross-compiled the *PolarSSL* library [2] for Android, an open source library similar to OpenSSL but more light-weight and easier to use and integrate. We then statically linked our PIN cracking program with the PolarSSL library as Android systems do not support dynamic linking out-of-the-box. Both the source code and the statically linked binary are available on our webpage under GPL v2 [36]. Again, you may use it independent of the FROST recovery image. An exemplary output can be found in Fig. 3.

### 3.3 Recovery Image and GUI

We integrated both the FROST LKM as well as our bruteforce program into an easy-to-use recovery image that allows to operate our system utilities with a few clicks. Technically, the FROST recovery image is based on the recovery image from *ClockworkMod*, a known provider of custom Android ROMs. A precompiled version of FROST for Galaxy Nexus devices can be found on our website. However, we believe that similar images can easily be build for a wider class of devices with the system utilities that we provide.



**Fig. 4:** Graphical user interface of the recovery image. F.l.t.r: recovered PIN from bruteforce, UI menu, and key recovery with the FROST LKM.

All in all, users can choose between one of the following options in the FROST recovery image:

- *Telephone encryption state*: To check the encryption state of the phone, we simply try to mount the userdata partition and check whether that fails.
- *Key recovery (quick search)*: Induces a quick search for AES keys as described in Sect. 3.1. Internally, the FROST LKM is loaded and its output is displayed to the user.
- *Key recovery (full search)*: Induces a full search for AES keys as described in Sect. 3.1. Also here, the FROST LKM is loaded.
- *RAM dump via USB*: To load full memory dumps to the PC we make use of the LiME module [21]. Similar to the FROST LKM, LiME parses a kernel structure to learn physical memory addresses. It then maps each physical page that is in system RAM, i.e., that does not belong to an I/O device, to a virtual address in kernel space. Each mapped page is transferred over a TCP socket to the computer via USB.
- *Crack 4-digit PINs*: Performs bruteforce attacks against weak PINs as described in Sect. 3.2.
- *Decrypt and mount /data*: Decrypts the user partition with recently recovered keys. To decrypt the userdata partition, we integrated a statically linked ARM binary of the *dmsetup* utility [43]. This option becomes available only if one of the key recovery methods or the bruteforce approach were successful, i.e., only after the decryption key is known.

## 4 On the Effectiveness of Cold Boot Attacks against Scrambled Telephones

We now give a detailed evaluation regarding the usability and the effectiveness of FROST. First we describe ways how to flash our recovery image onto phones in Sect. 4.1. For devices where that it is possible without destroying the user

partition, we look at the runtime performance of our key recovery and bruteforce procedures in Sect. 4.2. For other devices, we look at the information we can gain from RAM even though the user partition gets destroyed in Sect. 4.4. We also investigate the correlation of the operating temperature of Galaxy Nexus devices with the success of cold boot attacks in Sect. 4.3.

## 4.1 Flashing the Recovery Image

The question we answer in this section is, how do we cold boot a smartphone and install the FROST image when we just gained physical access to it? Or, how can law enforcement forensically investigate scrambled telephones with the help of FROST? An important point at the beginning is to assure that the device has enough power. Otherwise it should be charged first, because once a device loses power there is no possibility other than bruteforce to recover keys. Next, the device must be *cooled down* to increase the success rate of cold boot attacks (cf. Sect. 4.3). As a rule of thumb, we have positive experience with putting the device into a  $-15^{\circ}\text{C}$  freezer for 60 minutes. Take care to pack it up in a freezer bag due to water condensation.

After the phone has been charged and cooled down, we can perform cold boot attacks. First, we assume the bootloader of the device is already unlocked. Since the device has no reset button, we have to reboot it by unplugging the battery briefly. Shutting the device down in software from the lock screen is so slow that key information might get lost. To boot up the device quickly after the battery has been re-inserted, the power button must already be held *before* removing the battery. The entire process must happen quickly such that the phone is without power only for milliseconds. We recommend to practice this procedure several times before carrying it out in real cases.

Once the smartphone is up again, the risk of losing RAM contents is defeated. Flashing the recovery image does not destroy important RAM lines according to our tests. To flash the FROST image onto the phone, the buttons *Volume Up* and *Volume Down* must additionally be held during boot to enter the *Fastboot* mode. Once the phone is in Fastboot mode, it must be connected to a Linux PC via USB, and FROST can be flashed onto it with the command **fastboot flash recovery frost.img**. Afterwards, the *Recovery Mode* option must be selected from the phone menu in order to boot into our GUI, and to eventually recover keys.

If the bootloader is locked, we have additionally to run **fastboot oem unlock** before flashing the recovery partition. Unfortunately, this process does wipe all user data, and you have to confirm the warning: “To prevent unauthorized access to your personal data, unlocking the bootloader will also delete all personal data from your phone”. Once you confirm this warning, the scrambled user partition gets wiped. If that is the case, however, you can still use FROST to recover RAM contents (cf. Sect. 4.4); but you cannot decrypt the user partition anymore.

## 4.2 Runtime Performance

In this section we give a brief overview of the performance that you can expect when running FROST. The key recovery mode optimized for Galaxy Nexus (quick search mode) finishes in about  $9s$ . Contrary to that, the full search mode requires  $7m\ 40s$ . To take a full memory dump of 700 MB with help of the LiME module, we need  $3m\ 9s$ . 4-digit PINs are cracked within  $2m\ 58s$  at maximum, i.e., in one and a half minute on average. Although we only implemented the important 4-digit case, we can estimate that 5-digit PINs would be cracked in about  $15m$ , 6-digit PINs in  $2h\ 30m$ , and 7-digit PINs in about  $25h$ .

## 4.3 Impact of the DRAM Operating Temperature

We now analyze the reliability of FROST, i.e., the success rate of cold boot attacks, in dependence of the operating temperature and the time of battery removal. Earlier in our investigations, we recognized that the chance to recover AES keys from DRAM increases considerably if the phone is cold because it has just been switched on. We also experimented with putting the phone into a fridge and then into a freezer, and we got even better success rates. In the following we give exact benchmarks for this effect.

	$\varepsilon$	0.5 – 1s	1 – 2s	3 – 4s	5 – 6s
5 – 10 °C	0 (0%)	2 (0%)	1911 ( 5%)	8327 (25%)	24181 (73%)
10 – 15 °C	0 (0%)	976 (2%)	2792 ( 8%)	18083 (55%)	25041 (76%)
15 – 20 °C	0 (0%)	497 (1%)	4575 (13%)	20095 (61%)	25433 (77%)
20 – 25 °C	0 (0%)	421 (1%)	16461 (50%)	23983 (73%)	27845 (84%)
25 – 30 °C	1 (0%)	2204 (6%)	16177 (49%)	27454 (83%)	28661 (87%)

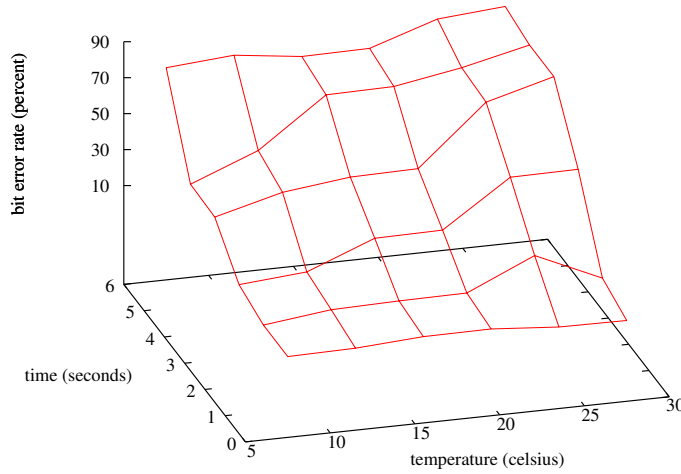
**Fig. 5:** Number of bit flipping errors per physical page (in total and percentage) dependent on the phone temperature and the time of battery removal.

Fig. 5 lists the bit error rate of RAM pages as a function of the device temperature and the time without power before reboot. To determine the device temperature we utilized an infrared thermometer and pointed it to the exactly same position of the phone’s motherboard each test run. To cool down the phone, we put it into a  $-15\text{ °C}$  freezer.  $25 - 30\text{ °C}$  is the normal operating temperature of a Galaxy Nexus,  $20 - 25\text{ °C}$  is reached after 10 minutes,  $15 - 20\text{ °C}$  after 20 minutes,  $10 - 15\text{ °C}$  after 40 minutes, and  $5 - 10\text{ °C}$  after 60 minutes inside the freezer.

To determine the bit error rate, we filled pages at fixed physical addresses entirely with  $0xff$ . The page size in Android is 4096 and hence, we filled them with  $4096 \cdot 8 = 32768$  one-bits. After rebooting the device, we re-considered the pages we recently filled and counted the bits that were zero. By this means, we

got the total number of inverted bits and we were able to estimate the overall bit error rate, as listed in Fig. 5.<sup>2</sup>

The most inaccurate part of our measurements are the times that a device was without power. The problem of rebooting a Galaxy Nexus is that battery removal is a manual, mechanic task and that milliseconds are crucial for the success of cold boot attacks. With  $\varepsilon$  we define the quickest unplugging-replugging procedure that we were able to perform; this was consistently below 500 ms. Thereover, we define four intervals up to six seconds, and say that we replugged the battery “somewhen” during these intervals. We explain inconsistencies of the table given in Fig. 5 mostly with inaccurate timings.



**Fig. 6:** Visualized bit error rate in dependence of time and temperatures.

In Fig. 6, we visualized the data set from Fig. 5. It becomes clear that the bit error rate of DRAM increases with both the temperature and the time without power. For example, at a temperatur of approximately 25 °C we have a bit error rate of already 50% after one second without power, whereas the corresponding bit error rate at temperatures around 10 °C is only 5%. Hence, besides replugging the battery quickly, putting a device into the freezer increases the chance of a successful cold boot attack considerably.

#### 4.4 Recovery of Non-Key Data from DRAM

We now assume that the bootloader of a Galaxy Nexus is locked and investigate which sensitive data we can forensically recover from DRAM without accessing

<sup>2</sup> Note that the highest possible bit error rate is 87.5%, and not 100%, because the passive state of 50% of RAM lines is 0xc0, and not 0x00.

the disk. We are after personal and corporate data such as contact lists, messaging, photos, and calendar entries.



**Fig. 7:** Photos we could recover from DRAM via cold boot attacks. The right most picture was taken with the Galaxy Nexus instantly before the attack. The other pictures were taken weeks before the attack with another smartphone, but they got synchronized with our target phone via Dropbox.

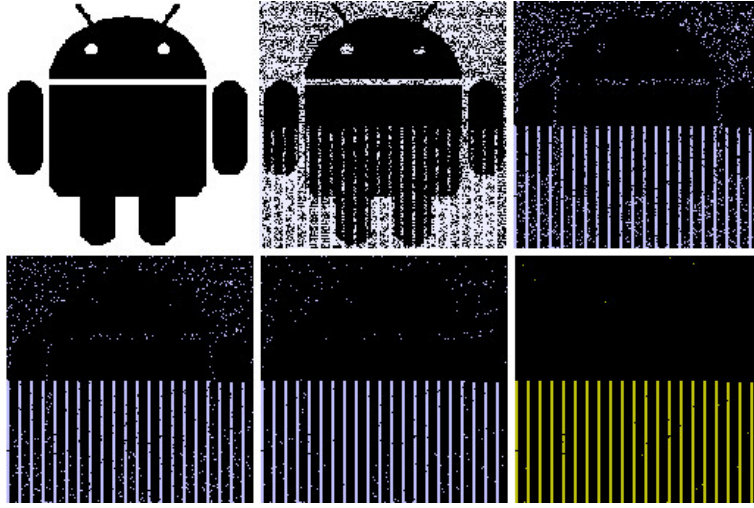
Technically, we used the FROST recovery image to take full memory dumps, as described above. We then examined the memory dumps with known system utilities like *strings* or *hexdump*, and made use of the program *PhotoRec* [16]. Besides photos, PhotoRec can recover websites, text files, databases, sound files, source codes, and binary programs from raw memory images. In our main test case, we were able to recover 68 JPEG and 199 PNG pictures, 36 OGG tracks, 295 HTML and 386 XML files, 215 SQLite databases, 28 ZIP and 105 JAR archives, 1214 ELF binaries, 485 JAVA source codes, and 6331 text files.

While most PNG images that we recovered were system images and logos, most JPEG files were personal photographs, as shown in Fig. 7. We were able to recover both pictures that were recently taken and older pictures. We were quite surprised as we first recovered pictures that were taken with another smartphone weeks before – the reason was that they got synchronized in background via Dropbox. In some cases, we recovered two variants of a photo: a small thumbnail and a bigger variant. In most cases, however, we could only retrieve smaller variants and high-resolution pictures remained secret.

We also recovered broken JPEG images with PhotoRec that were composed of several images and destroyed blocks. We conjecture that this effect comes from bit flipping errors from the cold boot attack, and so we visualized the influence of bit errors in Fig. 8. For the picture series in Fig. 8, we used a 4096-byte bitmap that exactly fits into one physical page.<sup>3</sup> We then increased the interval that the phone was without power during boot from 0.5 to 6 seconds. Whenever the bitmap header got destroyed, we fixed it manually in order to display the image. Fig. 8 impressively shows the effect and the distribution of bit errors. It also shows that the passive state of the first half of a physical RAM page is 0x00, while the passive state of the second half is 0xc0.

Most PNG images that we recovered were system files, but, for example, also the Wikipedia and Wikimedia Logos were available. Before the attack, we

<sup>3</sup> We used a bitmap rather than a JPEG in order to visualize bit errors because using JPEGs, entire blocks get destroyed rather than single bits.



**Fig. 8:** An Android bitmap after 0s, 0.5s, 1s, 2s, 4s, and 6s in DRAM without power. The cold boot attack has been deployed at room temperature.

intentionally surfed to *wikipedia.org*, and the HTML source of this page could be recovered, too. Even more critical, we found personal text files such as emails, and we found the entire Whatsapp chat history in RAM. We also explicitly searched for names of our contact list and we found each name to be present in RAM several times, as well as respective phone numbers and email addresses. Additionally, we recovered the birthday of some people from Jorte Calendar, and we even found passwords in RAM. Actually, we did search for the SSID of our WLAN and we could easily locate the according username and password in plaintext.

Concluding, we did recover the following sensitive information from RAM:

- New and old personal photos (from Dropbox).
- Recently visited websites.
- Emails and Whatsapp messages.
- Contact lists including names, phone numbers, and email addresses.
- Calendar entries like birthdays (Jorte Calendar).
- WLAN credentials: username and password.

Summarizing, we could recover dozens of sensitive information by simple means like the *strings* and *PhotoRec* utility. We did not use specialized forensic software such as the Volatility framework, and we still miss some interesting information like GPS coordinates and the list of recent phone calls. But we are confident that these information can be retrieved from RAM with more effort in future, too.

## 5 Countermeasures and Future Work

We believe our study is important for two reasons: First, it reveals a significant security gap in Android’s full disk encryption that must be counteracted in future. We present possible countermeasures in Sect. 5.1. Second, we provide the utility FROST which allows law enforcement to forensically recover telephones. We discuss future improvements for FROST in Sect. 5.2. We conclude in Sect 5.3.

### 5.1 Countermeasures

Physically loosing a smartphone, we can distinguish two severe consequences: (1) data loss or damage and (2) unauthorized access to data [28]. While the former can be counteracted by regular backup solutions today, the latter must be counteracted by *more secure* encryption solutions than thus that are available today. As we have shown, the current FDE system implemented in Android 4.0 is vulnerable to cold boot attacks – a generic threat that is known since years. All the more surprising it is, that no company attended to it in recent years. Linux’ dm-crypt as well as Microsoft’s BitLocker and Apple’s FileVault are still vulnerable. On the other hand, several academic projects have shown that cold boot resistant implementations of FDE are basically possible.

The common idea of these academic projects is simple: keep the key outside RAM. FrozenCache (2009) [22] holds the key in CPU caches, AESSE (2010) [26] holds it in SSE registers, TRESOR (2011) [27] in debug registers, and LoopAmnesia (2011) [29] in MSRs. All these solutions are Linux-based and thus, they are perfectly suited to get adopted for Android. Another solution would be to wipe the key from RAM each time the screen gets locked and to re-derive it from the PIN only when the screen gets unlocked. Apple’s iOS [5], for example, pursues this approach and, consequently, it is mostly immune to cold boot attacks.

### 5.2 Future Work

Our plans for future improvements of FROST are twofold. First, we want to make our recovery image available for more Android devices than just the Galaxy Nexus. However, we already provide device independent system utilities like the FROST LKM and the PIN cracking program on our webpage, such that experienced IT practitioners can compose recovery images for other devices on their own. To this end, we also provide appropriate howtos. The next device that we want to support ourselves is the Google Nexus 7 tablet manufactured by Asus.

Second, an important research topic for future releases is to find a way to retrieve physical memory dumps without the need to unlock the bootloader first. This task requires some electronic skills, but with available tools like the *RIFF Box* [33], it is feasible to retrieve RAM dumps via the JTAG interface already today. Another interesting question is how to acquire a full copy of the user partition without unlocking the bootloader. This might be possible with help of the RIFF Box and JTAG interface, too, or through other hardware



operations. At all events, it eventually would allow us to perform cold boot attacks against encryption keys although the bootloader is locked and thus, would have considerable consequences for practical scenarios.

### 5.3 Conclusions

The convenience that smartphones offer today for both private persons and companies cannot be ignored because they brought mobility, connectivity, and productivity to people. In fact, smartphones are miniaturized business computers and carry around personal data like emails, contact lists, messaging logs, and more. However, when loosing an Android-based smartphone chances are to lose confidential information although all possible security measures have been taken. Because, as we have proven, cold boot attacks are not restricted to x86 PCs but Android-based devices are equally affected.

To this end, we presented FROST, a forensic tool that can help law enforcement to recover encryption keys from scrambled telephones. We implemented both the cold boot approach and a brute-force approach. For the cold boot approach we implemented a FROST LKM that walks through all physical RAM pages and searches for AES key schedules. The biggest limitation to date, however, is that FROST requires an unlocked bootloader in order to break FDE. Otherwise, sensitive RAM contents can still be recovered but the disk remains inaccessible as it gets wiped when unlocking the bootloader.

### References

1. Mobile Threat Report Q2 2012. F-Secure Labs, Helsinki, Finland, Aug. 2012.
2. PolarSSL library - Crypto and SSL made easy. <http://polarssl.org>, Oct. 2012. version 1.2.0-pre1.
3. ANDREWS, W. The Smartphone and its Risks. RSM McGladrey Business Consulting, 2009.
4. ANDROID OPEN SOURCE PROJECT (AOSP). Notes on the implementation of encryption in Android 3.0. [source.android.com/tech/encryption/](http://source.android.com/tech/encryption/).
5. APPLE INC. iOS Security, 2012. Whitepaper.
6. AVIV, A. J., GIBSON, K., MOSSOP, E., BLAZE, M., AND SMITH, J. M. Smudge Attacks on Smartphone Touch Screens. In *WOOT '10 (4th USENIX Workshop on Offensive Technologies)* (Aug. 2010), Department of Computer and Information Science, University of Pennsylvania.
7. BECHER, M., DORNSEIF, M., AND KLEIN, C. N. FireWire - All your memory are belong to us. In *Proceedings of the Annual CanSecWest Applied Security Conference* (Vancouver, British Columbia, Canada, 2005), Laboratory for Dependable Distributed Systems, RWTH Aachen University.
8. BREAK & ENTER: IMPROVING SECURITY BY BREAKING IT. Adventures with Daisy in Thunderbolt-DMA-Land: Hacking Macs through the Thunderbolt interface, Feb. 2012.
9. CANNON, T., AND BRADFORD, S. Into the Droid: Gaining Access to Android User Data. In *DefCon '12* (July 2012), VIA Forensics.

10. CARBONE AND BEAN AND SALOIS. An in-depth analysis of the cold boot attack. Tech. rep., DRDC Valcartier, Defence Research and Development, Canada, Jan. 2011. Technical Memorandum.
11. CHIEN, E. Motivations of Recent Android Malware. Symantec Security Response, Culver City, California, 2011.
12. CHRISTOPHE DEVINE AND GUILLAUME VISSIAN. Compromission physique par le bus PCI. In *Proceedings of SSTIC '09* (June 2009), Thales Security Systems.
13. COVERT, A. iOS Encryption Is So Good, Not Even the NSA Can Hack It. gizmodo.com.
14. FIPS. Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, National Institute for Standards and Technology, Nov. 2001.
15. FRUHWIRTH, C. LUKS On-Disk Format Specification Version 1.1.1, Dec. 2008. Linux Unified Key Setup.
16. GRENIER, C. PhotoRec 6.13, Data Recovery Utility. <http://www.cgsecurity.org/wiki/PhotoRec>, Nov. 2011.
17. GUTMANN, P. Data Remanence in Semiconductor Devices. In *Proceedings of the 10th USENIX Security Symposium* (Washington, D.C., Aug. 2001), USENIX Association.
18. HALDERMAN, J. A., SCHOEN, S. D., HENINGER, N., CLARKSON, W., PAUL, W., CALANDRINO, J. A., FELDMAN, A. J., APPELBAUM, J., AND FELTEN, E. W. Lest We Remember: Cold Boot Attacks on Encryptions Keys. In *Proceedings of the 17th USENIX Security Symposium* (San Jose, CA, Aug. 2008), Princeton University, USENIX Association, pp. 45–60.
19. HENINGER, N., AND FELDMAN, A. AESKeyFind. <http://citp.princeton.edu/memory-content/src/>, July 2008.
20. JOANNA RUTKOWSKA. Evil Maid goes after TrueCrypt. <http://theinvisiblethings.blogspot.com/2009/10/evil-maid-goes-after-truecrypt.html>, Oct. 2009. The Invisible Things Lab.
21. JOE SYLVE. LiME - Linux Memory Extractor. In *ShmooCon '12* (Washington, D.C., Jan. 2012), Digital Forensics Solutions, LLC.
22. JÜRGEN PABEL. Frozen Cache. <http://frozenchache.blogspot.com/>, Jan. 2009.
23. KUMAR, M. Android facial recognition based unlocking can be fooled with photo. <http://thehackernews.com/>, Nov. 2011. The Hacker News.
24. MAYRHOFFER, R., AND KAISER, T. Towards usable authentication on mobile phones: An evaluation of speaker and face recognition on off-the-shelf handsets. In *Fourth International Workshop on Security and Privacy in Spontaneous Interaction and Mobile Phone Use (IWSSI/SPMU)* (Newcastle, UK, June 2012), University of Applied Sciences Upper Austria.
25. MICROSOFT CORPORATION. Windows BitLocker Drive Encryption: Technical Overview. Microsoft, July 2009.
26. MÜLLER, T., DEWALD, A., AND FREILING, F. AESSE: A Cold-Boot Resistant Implementation of AES. In *Proceedings of the Third European Workshop on System Security (EUROSEC)* (Paris, France, Apr. 2010), RWTH Aachen / Mannheim University, ACM, pp. 42–47.
27. MÜLLER, T., FREILING, F., AND DEWALD, A. TRESOR Runs Encryption Securely Outside RAM. In *20th USENIX Security Symposium* (San Francisco, California, Aug. 2011), University of Erlangen-Nuremberg, USENIX Association.
28. MUSLUKHOV, I. Survey: Data Protection in Smartphones Against Physical Threats. In *Term Project Papers on Mobile Security*. University of British Columbia, Apr. 2012.

29. PATRICK SIMMONS. Security Through Amnesia: A Software-Based Solution to the Cold Boot Attack on Disk Encryption. *CoRR abs/1104.4843* (2011). University of Illinois at Urbana-Champaign.
30. PONEMON INSTITUTE LLC. Smartphone Security: Survey of U.S. consumers. In *Ponemon Institute Research Report*. sponsored by AVG Technologies, Mar. 2011.
31. PONEMON INSTITUTE LLC. The Lost Smartphone Problem: Benchmark study of U.S. organizations. In *Ponemon Institute Research Report*. sponsored by McAfee, Oct. 2011.
32. POWER, R. McAfee Mobility and Security Survey: Dazzling opportunities, profound challenges. Carnegie Mellon CyLab, Santa Clara, California, 2011.
33. ROCKER TEAM. RIFF Box JTAG Revolution. <http://www.riffbox.org/>, 2010.
34. SAOUT, C. dm-crypt: a device-mapper crypto target, 2006. <http://www.saout.de/misc/dm-crypt/>.
35. SMITH, A. 35% of American adults own a smartphone. In *Pew Internet and American Life Project*. PewResearchCenter, July 2011.
36. STALLMAN, R., AND COHEN, J. GNU General Public License Version 2, June 1991. Free Software Foundation.
37. TEXAS INSTRUMENTS INCORPORATED. *OMAP 4 mobile applications platform*, 2009.
38. TRUECRYPT FOUNDATION. TrueCrypt: Free Open-Source On-The-Fly Disk Encryption Software for Windows 7/Vista/XP, Mac OS X and Linux. <http://www.truecrypt.org/>, 2012.
39. TURAN, M., BARKER, E., BURR, W., AND CHEN, L. Special Publication 800-132: Recommendation for Password-Based Key Derivation. Tech. rep., NIST, Computer Security Division, Information Technology Laboratory, Dec. 2010.
40. WRIGHT, S. The Symantec Smartphone Honey Stick Project. Symantec Corporation, Mar. 2012.
41. XDADEVELOPERS. GT-i9100 Galaxy SII FAQ. <http://forum.xda-developers.com>, Apr. 2011. Thread 1046748.
42. XDADEVELOPERS. Google Play Nexus not wiping after Bootloader Unlock. <http://forum.xda-developers.com>, Apr. 2012. Thread 1650830.
43. ZUGELDER, M. androidcrypt.py. <https://github.com/michael42/androidcrypt.py/>, Apr. 2012.