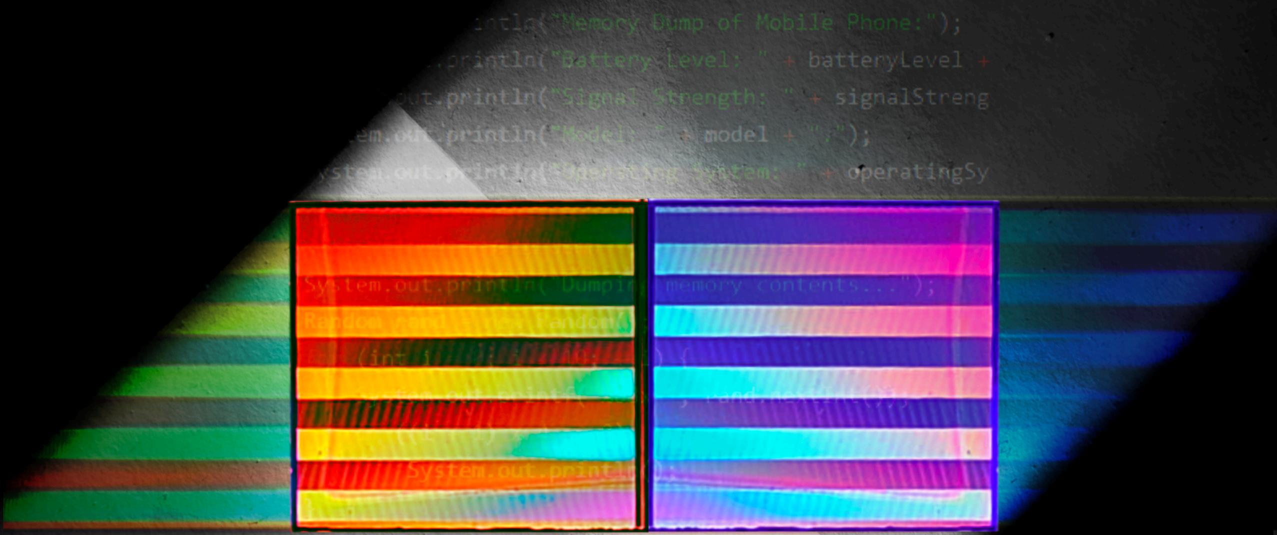
EFFECTIVE MOBILE FORENSICS THROUGH EXPLOITING THE MEMORY SECURITY

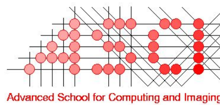


AYA FUKAMI

EFFECTIVE MOBILE FORENSICS THROUGH EXPLOITING THE MEMORY SECURITY AYA FUKAMI

Effective Mobile Forensics Through Exploiting the Memory Security

Aya Fukami



This work was carried out in the Netherlands Forensisch Instituut and in the ASCI graduate school. ASCI dissertation series number: 459
The printing of this thesis was financially supported by the Co van Ledden Hulsebosch Center, Netherlands Center for Forensic Science and Medicine.

Cover Design by: Maxim Smulders (Instagram: @Max1ms)
The concept behind the cover artwork symbolizes the notion that, with persistence in searching, you may eventually uncover what you seek. The front cover depicts the flash memory silicon die inside a chip, reflecting light. Searching in darkness is challenging, and there is rarely enough light to reveal everything clearly. However, the subtle illumination represents the ongoing effort to uncover what is hidden.

Copyright © 2024 by: Aya Fukami
ISBN 978-94-6496-233-8

Effective Mobile Forensics Through Exploiting the Memory Security

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. P.P.C.C. Verbeek
ten overstaan van een door het College voor Promoties ingestelde commissie,
in het openbaar te verdedigen in de Agnietenkapel
op woensdag dag 6 november 2024, te 10:00 uur

door

Aya Fukami

geboren te Miyazaki

Promotiecommissie

Promotores:	prof. dr. ir. C.T.A.M. de Laat	Universiteit van Amsterdam
	prof. dr. ing. Z.J.M.H. Geradts	Universiteit van Amsterdam
Copromotores:	dr. F. Regazzoni	Universiteit van Amsterdam
Overige leden:	prof.dr. F. Freiling	Friedrich-Alexander-Universität Erlangen-Nürnberg
	dr. T. Souvignet	Université de Lausanne
	prof. dr. ir. H.J. Bos	Vrije Universiteit Amsterdam
	prof. dr. C. Schaffner	Universiteit van Amsterdam
	prof. dr. P. Grosso	Universiteit van Amsterdam
	dr. K. Papagiannopoulos	Universiteit van Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Contents

Abstract **i**

Samenvatting **ii**

Acronyms **iii**

1 Introduction **1**
- 1.1 Research Questions . . . 2
- 1.2 Key Contributions . . . 2
- 1.3 Thesis Overview and the Sources of the Chapters . . . 3
- 1.4 Research Material Availability . . . 5

2 Evolution of Mobile Forensics: From Physical Data Analysis to Breaking Security Barriers **7**
- 2.1 Challenges and Advances in Mobile Forensics Against Increasing Security Measures . . . 7
- 2.2 Paradigm Shift in Mobile Forensics . . . 9
- 2.3 Mobile Forensic Techniques in the Encryption Era . . . 12
- 2.4 Mobile Forensics Through Non-Volatile Memory Storage . . . 15
- 2.5 Conclusion . . . 17

3 Effective Forensic Data Recovery from Flash Memory **19**
- 3.1 Challenges in NAND Flash Memory Forensics and Mitigation Techniques for Data Recovery . . . 19
- 3.2 Multi-Level Cell NAND Flash Memory Basics . . . 22
- 3.3 Bit Errors in Flash Memory During Forensic Analysis . . . 25
- 3.4 Read Retry Features in Flash Memory . . . 28
- 3.5 Experimental Evaluation of Enhancing Reliability in Chip-off Analysis through Read Retry . . . 30
- 3.6 Conclusion . . . 40

4 Data Recovery from Embedded MultiMediaCard **41**
- 4.1 Embedded MultiMediaCard in Mobile Devices and Its Impact in Forensic Data Extraction . . . 41
- 4.2 Embedded MultiMediaCard Structure . . . 43
- 4.3 Extracting Data from Flash Memory in Embedded MultiMediaCard . . . 48
- 4.4 Data Recovery from Embedded MultiMediaCard . . . 52

4.5 Implications and Extensions of Embedded MultiMediaCard Data Recovery in Digital Forensics 57
4.6 Conclusion 60

5 Exploiting Replay Protected Memory Block Authentication in Trusted Execution Environment **61**
5.1 Replay Protected Memory Block 61
5.2 Dissecting the Use of Replay Protected Memory Block on an Android Device 63
5.3 Restoring Android Device State by Exploiting Replay Protected Memory Block 70
5.4 Attack Mitigation 73
5.5 Conclusion 75

6 Bypassing Replay Protected Memory Block Authentication Through Fault Injection **77**
6.1 Fault Injection 77
6.2 Experiment Setup 78
6.3 Characterizing the Target Against Fault Injection 80
6.4 Glitching Replay Protected Memory Block Authentication 85
6.5 Discussions 90
6.6 Conclusion 92

7 Conclusion **93**
7.1 Addressing the Research Questions 93
7.2 Future Research 95

Bibliography **97**

Acknowledgements **109**

Effective Mobile Forensics Through Exploiting the Memory Security

Abstract

Mobile phones have become an indispensable part of our lives, and contain a wealth of information that can be useful in criminal investigations. However, extracting this information from modern smartphones can be a challenge for forensic investigators.

This research delves into the hardware vulnerabilities of memory devices in modern smartphones, which represent potential avenues for bypassing security mechanisms. By identifying and exploiting these vulnerabilities, this thesis demonstrates that effective strategies for extracting plaintext data from secured devices, thereby overcoming some of the limitations imposed by conventional forensic methods.

In-depth analysis is presented on the development of novel forensic techniques that leverage flaws in memory security architectures to access encrypted or otherwise protected information. These methods not only highlight the potential for accessing valuable data but also address the broader implications of device security for forensic investigations.

This study emphasizes the importance of continuous advancements in forensic science to keep pace with evolving security technologies in mobile devices. It advocates for a proactive approach to forensic research, one that anticipates future security measures and develops corresponding countermeasures to ensure the accessibility of critical evidence in legal settings.

Effective Mobile Forensics Through Exploiting the Memory Security

Samenvatting

Mobiele telefoons zijn een onmisbaar onderdeel van ons leven geworden en bevatten een schat aan informatie die nuttig kan zijn bij strafrechtelijke onderzoeken. Het extraheren van deze informatie uit moderne smartphones kan echter een uitdaging zijn voor forensisch onderzoekers.

Het onderzoek verdiept zich in de hardwarekwetsbaarheden van geheugenapparaten in moderne smartphones, die potentiële manieren vormen om beveiligingsmechanismen te omzeilen. Door deze kwetsbaarheden te identificeren en te exploiteren, toont dit proefschrift effectieve strategieën voor het extraheren van platte tekstgegevens van beveiligde apparaten, waarmee enkele beperkingen worden overwonnen die worden opgelegd door conventionele forensische methoden.

Er wordt een diepgaande analyse gepresenteerd van de ontwikkeling van nieuwe forensische technieken die gebruikmaken van gebreken in geheugenbeveiligingsarchitecturen om toegang te krijgen tot gecodeerde of anderszins beschermde informatie. Deze methoden benadrukken niet alleen het potentieel voor toegang tot waardevolle gegevens, maar pakken ook de bredere implicaties van apparaatbeveiliging voor forensisch onderzoek aan.

Deze studie benadrukt het belang van voortdurende vooruitgang in forensische wetenschap om gelijke tred te houden met de evoluerende beveiligingstechnologieën in mobiele apparaten. Het pleit voor een proactieve benadering van forensisch onderzoek, waarbij rekening wordt gehouden met toekomstige veiligheidsmaatregelen en passende tegenmaatregelen worden ontwikkeld om de toegankelijkheid van cruciaal bewijsmateriaal in juridische omgevingen te waarborgen.

Acronyms

ADB Android Debug Bridge. 72

AP Application Processor. 66

BCH Bose Chaudhuri Hocquenghem. 20, 33

BGA Ball Grid Array. 43

DFU Device Firmware Update. 14

ECC Error Correction Code. 20, 25, 33, 36, 38, 39, 45, 46

EDL Emergency Download. 14

EL Exception Level. 64, 72

eMCP Embedded Multi Chip Package. 42, 67, 68, 71, 75

eMMC embedded MultiMediaCard. 2, 16, 41–62, 67, 70–75, 77–80, 82, 87, 91, 94, 95

FBE File Based Encryption. 11

FDE Full Disk Encryption. 11, 64, 66

FPGA Field Programmable Gate Array. 30

FTL Flash Translation Layer. 45

HMAC Hash Message Authentication Code. 62, 66, 67, 85–88

IC Integrated Circuit. 10, 15, 42–44, 50, 58

IoT Internet of Things. 42, 58

ISP In-System Programming. 9

ISPP Incremental Step Pulse Programming. 24

JEDEC Joint Electron Device Engineering Council. 42, 57, 61, 74, 81, 86

JTAG Join Test Action Group. 9, 13

LDPC Low Density Parity Check. 20

LFSR Linear Feedback Shift Register. 45, 46

LSB Least Significant Bit. 23, 24

MLC Multi Level Cell. 22, 23, 25, 30, 70

MMU Memory Management Unit. 64

MSB Most Significant Bit. 23, 24

NIST National Institute of Standards and Technology. 9

NVMe Non-volatile Memory express. 16, 95

OS Operating System. 11, 16, 67

PBL Primary Boot Loader. 13

PCB Printed Circuit Board. 9, 12, 20, 49, 52, 71, 79

QSEE Qualcomm Secure Execution Environment. 64–67, 72, 74

RBER Raw Bit Error Rate. 30, 32, 33, 35–39

REE Rich Execution Environment. 64, 65

ROM Read Only Memory. 13

RoT Root of Trust. 11, 12, 14, 15

RPMB Replay Protected Memory Block. 3, 6, 17, 61–64, 66–68, 70–75, 77, 78, 85–92, 94, 95

SBL Secondary Boot Loader. 13

SCA Side Channel Analysis. 15

SEM Scanning Electron Microscope. 10

SEP Secure Enclave Processor. 11

SLC Single Level Cell. 70

SMC Secure Monitor Call. 65

SoC System on a Chip. 12–15, 18, 63, 66, 67, 73, 75

SSD Solid State Drive. 19, 42

TEE Trusted Execution Environment. 11, 61, 62, 64, 65, 91

UFS Universal Flash Storage. 16, 42, 61, 95

uMCP UFS-based Multi Chip Package. 16, 42

0

1 Introduction

Mobile forensics is a specialized area of forensic science that applies scientific methods to extract data from mobile devices, which can often serve as critical evidence in court. The most commonly used mobile devices today are smartphones, typically locked with user passwords. Historically, performing a physical data dump of a device has been considered the last resort in mobile forensics, as it allows examiners to access all data, including deleted files. However, with encryption now being a default feature on major smartphones, physical dumps alone no longer yield meaningful results. Additionally, the complexity of encryption schemes and secure computing layers unique to each smartphone model makes decrypting physical dumps an exceptionally challenging task. Consequently, digital forensics research has increasingly shifted towards exploiting software vulnerabilities in the target system, leading to a decline in memory-focused research. Despite this trend, there remains unexplored research potential in memory storage devices. In addition to continuously evolving toward faster and denser technology, the memory devices used in modern smartphones are becoming more complex, integrating their dedicated controllers inside a single package, which provides advanced security features. These evolving architectures require in-depth analysis for effective forensic investigations. By focusing on the vulnerabilities in these memory devices, new opportunities for advancing mobile forensic techniques can be explored.

1.1 Research Questions

Throughout this thesis, the following research question is explored: **"What kind of security features in flash memory can be exploited to perform effective data extraction from modern mobile devices?"** This question is addressed through the following sub-questions:

- **What are the current challenges in forensic data extraction from mobile devices?**
  The current status of mobile forensics is reviewed in Chapter 2, addressing the evolving challenges in extracting data from mobile devices.
- **How is the mobile device data stored in flash memory, and how should it be extracted?**
  The hardware architecture of flash memory and embedded MultiMediaCard (eMMC) is examined in Chapter 3 and Chapter 4, respectively, as these are widely used memory systems in modern mobile devices. The structural architecture of these devices is investigated, followed by experimental data recovery.
- **Which security features of flash memory are used in modern mobile device to secure the data? And how can they be exploited?**
  A detailed analysis of eMMC is conducted, focusing on security features such as secure data erasure and replay attack protection. Reverse engineering of these security features, both at the hardware and software levels, is carried out to exploit them for forensic data recovery in Chapter 5 and Chapter 6.

1.2 Key Contributions

This thesis mainly makes the following contributions:

- **Review of the evolution of mobile forensic techniques:**
  As smartphones and IoT devices have become integral to daily life, their security mechanisms and regulatory frameworks have grown increasingly complex. This thesis critically reviews the effectiveness of current standard mobile forensic methods in light of these developments.
- **Comprehensive hardware analysis of modern memory devices:**
  NAND flash memory, due to to its cost-efficiency, is widely adopted as storage media in contemporary digital devices. This research conducts an in-depth investigation into its architecture and identifies vulnerabilities that can be leveraged for digital forensic purposes.

- **Reverse-engineering of security features in memory devices:**
  Modern managed flash memory incorporates security features such as Secure Erase and the Replay Protected Memory Block (RPMB). This thesis reverse-engineers these features to assess their practical implementation, revealing that they are not entirely secure and can be exploited for effective forensic data recovery.
- **Exploitation of memory security vulnerabilities for enhanced mobile forensics:**
  Despite the implementation of security mitigation measures in modern memory devices, this research uncovers vulnerabilities in both hardware and software. Exploiting these weaknesses allows for more effective forensic data extraction from mobile devices, overcoming certain security barriers.

1.3 Thesis Overview and the Sources of the Chapters

Each chapter of this thesis is based on research work published in various academic outlets over the years. Below is a list of the publications along with their abstracts.

- Chapter 2
  A. Fukami, R. Stoykova, and Z. Geradts, "A new model for forensic data extraction from encrypted mobile devices," ***Forensic Science International: Digital Investigation*****. Volume 38, 2021.**
  The authors explained the increased encryption and security protection measures in modern mobile devices and their impact on traditional forensic data extraction techniques for law enforcement purposes. It was also demonstrated that in order to overcome encryption challenges, new mobile forensic methods rely on bypassing the security features and exploiting system vulnerabilities. A new model for forensic acquisition was also proposed, being supported by a legal framework focused on the usability of digital evidence obtained through vulnerability exploitation.
  K. Schot and A. Fukami, "In-Situ Global Ultra Thinning of Live Chip Backside for Digital Forensic and Failure Analysis," ***Proceedings of the 49th International Symposium for Testing and Failure Analysis*****. pp. 205-208, 2023.**
  The authors presented a new backside thinning techniques of a system-on-a- chip (SoC) while preserving its packaging integrity on a printed circuit board within a smartphone. The authors effectively achieved comprehensive thinning of bulk Silicon side of a SoC with more than 100 mm^2 surface area to a sub-10µm thickness.
- Chapter 3
  A. Fukami, S. Ghose, Y. Luo, Y. Cai, and O. Mutlu, "Improving the reliability of chip-off forensic analysis of NAND flash memory devices," ***Digital Investigation*****, Volume 20, 2017.**

The authors performed an analysis of the errors introduced into multi-level cell (MLC) NAND flash memory chips after the device has been seized. It was identified that a large number of bit errors can be introduced to the flash memory by a long storage time and thermal-based chip removal in digital forensic procedure. The authors successfully reduced the errors through the hardware voltage control called read-retry.

J. P. van Zandwijk and A. Fukami, "NAND Flash Memory Forensic Analysis and the Growing Challenge of Bit Errors," *IEEE Security & Privacy*, Volume 15, pages 82-87, 2017.

The authors discussed challenges imposed by reliability aspects of modern NAND-flash memory chips from a digital forensic perspective, and describe how acquisition and analysis techniques can be adapted to recover accurate and relevant data from NAND-flash memory chips. The authors also describe the idea of using error information from NAND flash memory chips as a means to infer forensically relevant information about the device, such as the age of the stored ata.

- Chapter 4

A. Fukami, S. Sheremetov, F. Regazzoni, Z. Geradts and C. De Laat, "Experimental Evaluation of e.MMC Data Recovery," *IEEE Transactions on Information Forensics and Security*, Volume 17, pp. 2074-2083, 2022.

Embedded Multimedia Cards (eMMCs) data recovery procedures were explored in this article. The authors investigated inside structures of eMMCs, and evaluate advanced data recovery procedures. It was discovered that data can be recovered, sometimes more than 99%, even after it is erased through eMMC Secure Erase routine.

A. Fukami, F. Regazzoni and Z. Geradts, "Data Sanitization on eMMCs," *28th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 455-460, 2023.

The authors analyzed repurposed eMMCs mounted on digital devices, and evaluated their sanitization practice. It was found that the data from the formerly used device can still be recovered, which may lead to an unintentional leakage of sensitive data such as personally identifiable information (PII). Authors discussed the difference between the traditional NAND flash memory and the eMMCs, and reviewed how the data sanitization schemes should be introduced to the eMMCs.

- Chapter 5

A. Fukami, R. Buurke and Z. Geradts, "Exploiting RPMB authentication in a closed source TEE implementation," *Forensic Science International: Digital Investigation*, 2024.

The usage of the Replay Protected Memory Block (RPMB) in an eMMC mounted on a smartphone was reverse-engineered both through software and the hardware. The authors achieved in extracting the secret key, which allows to manipulate the RPMB data of the target device. By exploiting the poor implementation of the

RPMB of the target device, the authors succeeded in breaking the anti-rollback protection to enable data restoration after the data wipe operation has been completed.

- Chapter 6
  A. Fukami and R. Buurke, "Keyless Entry: Breaking and Entering eMMC RPMB with EMFI" ***ACM Wisec 2024*****, 2024.** The authors performed the electromagnetic fault injection (EMFI) against the RPMB authentication in eMMCs. By identifying the most vulnerable location of a target eMMC against the EMFI, the authors achieved in bypassing the cryptographic authentication and writing the arbitrary value in the protected RPMB in eMMCs without knowledge of the secret key.

1.4 Research Material Availability

Data and other related resources used for this thesis are available at
Chapter 2: `https://github.com/topig/Flash_Multi_Write_Test`
Chapter 3: `https://github.com/topig/RPi-NAND`
Chapter 6: `https://github.com/topig/RPMB_Glitching`
Other image data can be available on request.

1

2

Evolution of Mobile Forensics: From Physical Data Analysis to Breaking Security Barriers

In modern criminal investigations, mobile devices are frequently seized from crime scenes, and the data they contain often serves as critical evidence. Over the decades, numerous mobile forensic techniques have been developed and rigorously evaluated through research to extract potential evidence from these devices. However, as mobile devices become essential tools for daily life, security and privacy concerns grow, and modern smartphone vendors have implemented multiple types of security protection measures to guard against unauthorized access to the data on their products. This trend makes forensic acquisition harder than before. As a result, data extraction from those devices for criminal investigation is becoming a more challenging task. Today, mobile forensic research focuses on identifying more invasive techniques, such as bypassing security features and breaking into target smartphones by exploiting their vulnerabilities. In this chapter, increased encryption and security protection measures in modern mobile devices are explored. Changes in digital forensic techniques according to those security features will be discussed.

2.1 Challenges and Advances in Mobile Forensics Against Increasing Security Measures

Forensic analysis of mobile devices for criminal investigation has become an increasingly critical investigative capability for law enforcement agencies. In recent decades, various

researchers in forensic science have established methods and processes to extract evidence data from mobile devices in a forensically sound manner [1, 2, 3]. Those methods have been widely used for forensic purposes in real cases, and have tackled general challenges in mobile forensics, such as the lack of standardization within the mobile industry and the rapid rate at which mobile device technology changes. However, on the other hand, new challenges have recently been imposed by the strong security features in modern mobile devices [4]. Encryption, together with other security guard features, has clearly created challenges for forensic investigators seeking to extract data from mobile devices seized at crime scenes. Those security features have disabled many of the data acquisition methods that have been used historically, and new methods for acquiring data from modern mobile devices must be explored.

The challenges posed by encryption were publicly highlighted during the 2015 dispute between Apple and the FBI following the widely reported San Bernardino, California, terrorist attack. That case not only sparked an intense legal debate about the regulation of cryptography and governmental access to encrypted devices, but also brought public attention to issues around the security and privacy of data stored on personal mobile devices. Not surprisingly, mobile device vendors have been implementing higher levels of security features in their products to address personal data protection. Currently, on modern mobile devices, user data is usually highly secured by default from malicious access by unauthorized attackers.

The impact of encryption on forensic analysis, as well as effective data acquisition processes, has been widely researched in the computer forensics domain [5, 6, 7, 8]. It has been suggested that temporary files, data on volatile memory, metadata of the encryption scheme, or access to the key management system can decrypt the target data, thereby allowing examiners to extract original data, which can then be used for criminal investigations. However, challenges in data acquisition from encrypted mobile devices come from the fact that those pieces of listed data are not accessible by default, requiring modification of the exhibit device or complicated reverse-engineering at multiple layers in software and hardware.

Although modifications to the exhibit device may be necessary, it is imperative to uphold the most important principle of digital forensics: "digital evidence must remain unaltered." Failure to adhere to this principle could compromise the integrity of the evidence, thereby diminishing its value in the court trial process.

2.2 Paradigm Shift in Mobile Forensics

2.2.1 Traditional Mobile Forensic Techniques

The acquisition techniques used in mobile forensics have been categorized using the classification system suggested by National Institute of Standards and Technology (NIST). The classification system includes the following five levels [9, 4]. A brief summary of the technical methods used at each level is explained below, along with the challenges that these techniques face when analyzing modern mobile devices:

- Level 1: Manual Extraction
  An examiner directly manipulates the target mobile device using the device's input interface (i.e., keypads and buttons), and records the content shown on the display of the device. As long as the target device is in unlocked state and it takes users input, this is still a valid process to extract data from modern digital devices. However, given the large volume of data stored in the recent devices, this process is not always feasible.

- Level 2: Logical Extraction
  There are various off-the-shelf tools available that allow users to connect a target device through external interfaces, such as USB or Bluetooth, and extract existing files. This process is effective if the target device is unlocked and the examiner has the necessary privileges to access the data. While individual files, such as photos, can be accessed with standard user privileges, system files and databases used by messaging applications typically require higher privileges. Additionally, modern applications often include "burn-time" features, where data, such as received messages, are erased after a set period. Therefore, forensic examiners must consider timing carefully after seizing the device.

- Level 3: Hex Dumping / JTAG
  When techniques at this level are employed, the full or partial raw data (hex dump) stored in the storage media of the target mobile device is acquired. Hex dumping is performed using available access ports on the target's Printed Circuit Board (PCB), such as Join Test Action Group (JTAG) or In-System Programming (ISP) [10]. Techniques that can acquire raw data without hardware destruction are generally classified within this category. The acquired raw data must then undergo further processing to be converted into human-readable form. However, since PCB layouts are proprietary to manufacturers, identifying these access ports is a challenging task. Additionally, debug interfaces are often disabled on modern digital devices, further complicating hex dumping for forensic examiners.

- Level 4: Chip-off
  Chip-off is a technique in which the non-volatile memory component is detached

from the main circuit board [11, 12]. The physical data stored in the memory chip is then directly extracted for further analysis. Through this method, an examiner can obtain an identical copy of the entire raw data of the target mobile device, including the data stored in unallocated areas, which possibly contain remnants of deleted data. For modern mobile devices, as the physical data is typically encrypted, further processing is required in order to recover the human-readable data after extracting data through chip-off.

- Level 5: Micro Read
  Micro read is a technique where the die in an Integrated Circuit (IC) chip is exposed from its package and the electrical status of the target circuit is directly observed through a Scanning Electron Microscope (SEM). This technique is highly destructive, as the internal structure of an IC chip needs to be exposed by removing its packaging material and bulk silicon. Furthermore, a highly dedicated lab is required for this level of analysis since multiple specialized equipment such as SEM and the decapsulation equipment are required to perform Micro Read analysis.

Data acquired through Level 1 and 2 techniques is typically referred to as logical data, while data obtained via Level 3 to 5 techniques is known as physical data, which has the advantage of including remnants of deleted data. Generally, data parsing is necessary to render physical data into human-readable form. In traditional mobile forensic models, it is commonly understood that higher acquisition levels increase the likelihood of successful forensic data recovery. As examiners employ higher acquisition levels, the range of accessible data expands. Additionally, physical acquisition methods can bypass user authentication mechanisms, such as PIN codes and passwords, and do not require the target device to be in a normal booting state. Consequently, law enforcement agencies widely adopt chip-off data acquisition as the highest-level data extraction technique for various mobile devices. Although micro-read is ranked as the highest level in the aforementioned classification system, and past research has demonstrated that reading data directly from the memory die is possible [13], it is not considered a practical mobile data extraction technique in mobile forensics, to the best of the author's knowledge. This is primarily due to the shrinking technology size in semiconductor fabrication and the increasing density of memory storage.

2.2.2 Encryption and Other Security Features in Modern Mobile Devices

In order to protect user privacy and provide confidentiality of data, encryption techniques are currently implemented in modern mobile devices by default. Traditionally, in mobile devices, encryption techniques were applied at the application level in order to protect individual user data such as emails and photos. With the growing concerns over security and

privacy, however, encryption techniques are now implemented at the Operating System (OS) level with hard-coded unique key information which is not accessible even by device manufacturers. Therefore, mobile device data at rest is stored in an encrypted manner. Two types of encryption schemes are frequently used in mobile devices. One is Full Disk Encryption (FDE) and the other is File Based Encryption (FBE) [14]. FDE is a technique where the entire user data partition is encrypted with a single encryption key, while FBE encrypts different files with different keys, allowing files to be decrypted independently. In Apple devices, FDE was first introduced in iPhone 3GS with iOS 3.X [15]. Apple devices with iOS versions higher than 8 use FBE which encrypts user data on a per file basis with a user passcode. In Android devices, FDE was introduced in Android 4.4, and was supported up until Android 9. Starting with Android 7.0, FBE has been used as the standard encryption technique. As of 2020, it is reported that more than 80 percent of the Android devices on the market are running on an Android version higher than 6 [16]. This means that user data in the Android devices that are seized during the criminal investigation is now mostly encrypted.

In addition to the encryption techniques, other "security by design" features are implemented in modern mobile devices. One example is Root of Trust (RoT). When a mobile device boots, each hardware and software component in the boot-chain is validated to ensure that only the legitimate components would run on the system. If the validation fails due to unsigned software or for other reasons, the target device does not boot, denying access to the device by malicious users. This makes several mobile forensic methods such as using special bootloaders suggested by Vidas et al. unworkable [17] . Additionally, the secure mode of main processor called the Trusted Execution Environment (TEE) is heavily utilized in modern smartphones. TEE provides an isolated environment for security critical components in a system by separating the secure operating system from a normal operating system, both running on the same hardware device. Hence a secure world and a normal world can co-exist on a system. ARM's TrustZone technology is largely used in Android devices. Apple uses a similar technology called Secure Enclave Processor (SEP) for isolating the cryptographic key and other sensitive information processing. If the TEE is used, even "rooting", or acquiring the highest privilege in the system does not allow access to the encryption key-related data. By including those security features, mobile device manufacturers are protecting not only user data, but also their corporate proprietary data and technologies. As a result, users have little freedom to control their own mobile devices, and they are limited to using them within the device or the OS vendor's closed ecosystem.

2.3 Mobile Forensic Techniques in the Encryption Era

Given the complex structure of modern smartphones, forensic data extraction from these devices requires a combination of different techniques. Some of the main techniques currently in use are listed below. Often, multiple processes are necessary to access and present user data stored on smartphones in a readable format.

2.3.1 Reverse Engineering

The core components of modern mobile devices consist of a System on a Chip (SoC), memory, and sometimes an additional dedicated security chip. Extracting meaningful data from these devices often requires reverse engineering both the hardware and the software running on each component. When an RoT is implemented, the initial code executed on the device is hard-coded into the processor via one-time-programmable memory, and device-specific keys are often embedded directly into the processor. As a result, chip-level reverse engineering becomes a valuable method for accessing this critical information. In 2023, research by Schot and Fukami [18] demonstrated a technique to remove bulk silicon and access the SoC logic while maintaining the SoC's operational status within a mobile phone. By employing reactive ion etching, bulk silicon was thinned while keeping the chip functional on the Printed Circuit Board (PCB). This SoC-level reverse engineering approach, comparable to Micro Read methods, enables the retrieval of secret information directly from the SoC.

On the software side, the code running on modern mobile devices is not always accessible to third parties. While leaked code may sometimes be available, otherwise, one can attempt to extract it by injecting faults into the secured system. Once the code is obtained, reverse engineering can be employed to understand the structure of the target security system.

2.3.2 Vulnerability Exploitation

After reverse engineering the components running on the target device, investigators may uncover system vulnerabilities that can be exploited to execute arbitrary code. Once these vulnerabilities are identified, forensic investigators can attempt to run arbitrary code to brute-force the password or extract encryption key-related information. By acquiring the secret information, investigators can attempt to decrypt the encrypted data stored on the target device. Security researchers continuously search for these vulnerabilities in consumer devices, while smartphone manufacturers regularly update their software and firmware

over the air to patch known vulnerabilities and protect their products. However, if a target device has not been updated, known vulnerabilities may still be exploitable.

Another way to break the chain of trust is by injecting faults or glitches into the target device through hardware. Hardware fault injection methods include underfeeding the power supply, transmitting electromagnetic signals, and injecting optical beams. These techniques are used to induce unintended behavior in the target device. By injecting a fault into the code verification scheme of the boot chain, it is possible to execute arbitrary code on the device. Research has already demonstrated the effectiveness of fault injection for compromising the boot sequence and executing code with the highest privileges from an Android device [19]. Fault injection can also be useful for disabling the lock on debugging interfaces, such as JTAG, on the target device.

2.3.3 Utilizing Custom Bootloaders

If an examiner can load a custom bootloader into the target device during the boot process and run it, there is a great chance that the device can be manipulated by running arbitrary code, making physical data acquisition possible. Traditionally, loading a custom bootloader was enabled by the device manufacturer. Special modes (i.e., download mode or rescue mode) allowed users to run a custom bootloader on the target system during the boot-up. In modern devices, however, in order to maintain system integrity, manufacturers enable bootloaders to run only after they are properly verified to be signed, allowing only their codes to run on the device. Bootloaders are responsible for initializing hardware components and loading the operating system, which then starts device operations, including encryption. When a modern mobile device is powered on, multiple bootloaders are executed in sequence. The first bootloader, hard-coded into the Read Only Memory (ROM) of the application processor, is known as the boot ROM or Primary Boot Loader (PBL). This first boorloader then loads the Secondary Boot Loader (SBL), which typically loads another bootloader responsible for finally loading the operating system [20]. Bootloaders are only loaded into system memory once the verification processes have been successfully completed, allowing the system to proceed with normal boot operations. The loading of bootloaders via download mode occurs at the SBL level. Verification processes run in a chain, meaning that code in each boot stage can only execute if its signature is verified by the preceding stage. This process begins with the initial verification key, which is stored in the one-time-programmable memory area of the SoC, ensuring the key cannot be tampered with. In the Android security model, the bootloader verifies the integrity of the code

by using a public key stored on the device. The code itself is signed with a private key that is not present on the device, but the corresponding public key is used to verify the signature. The hash of this public key, which in this case is a RoT, is stored in hardware fuses within the SoC. During the boot process, the bootloader compares the public key hash with the value stored in the fuses. If the signature is verified successfully, the boot process continues; otherwise, the device is halted to prevent unauthorized code from running.

For some models of modern mobile devices, signed bootloaders may be publicly available [20]. By flashing these bootloaders, which may contain known vulnerabilities, onto the target smartphone, an examiner can potentially gain the highest privileges on the device, leading to full control and successful acquisition of memory data. Additionally, if anti-rollback mechanisms are not implemented, the examiner can attempt to downgrade components of the boot chain to earlier versions. This approach allows exploitation of vulnerabilities that have been addressed in security updates of newer boot chain versions. However, the most effective method for gaining access and executing arbitrary code is to exploit vulnerabilities in the boot ROM, a technique that has been researched and employed for data access on modern mobile devices [21].

Although modern mobile devices prohibit users from loading custom bootloaders, it is now widely known that boot ROM-level flashing is possible by booting the device into the processor-level special boot mode. The terminology for these boot modes varies by manufacturer. For example, Qualcomm chipsets use Emergency Download (EDL) mode, Apple chipsets use Device Firmware Update (DFU) mode, and MediaTek chipsets use Brom (BootROM) or PreLoader mode. These modes allow mobile phone manufacturers to flash software onto their devices. Forensic examiners can utilize these modes to flash customized bootloaders onto the target smartphone, which facilitates the acquisition of user data without altering it. Unless additional authorization mechanisms are in place, entering these special modes typically requires a specific set of commands, a special cable, or hardware modifications. The use of custom bootloaders for data acquisition is gaining popularity because this technique can be applied to a wide range of devices with the same chipset. Additionally, it is often challenging for manufacturers to patch vulnerabilities at the processor level. Research has demonstrated that vulnerabilities in the bootloader of popular chipsets can be exploited for user data acquisition [20, 22].

2.3.4 Side-Channel Analysis

The hardware-bound key stored in the one-time-programmable memory of the SoC is crucial for protecting the overall security of modern mobile devices. One application of this key is to derive the necessary information for decrypting stored data on a mobile device. Another application is to verify the integrity of the code running on the device. Therefore, obtaining this key would provide an examiner with the ultimate opportunity to access user data, as it is essential for breaking the RoT or recovering other key information. Given that this hardware-bound key is highly secure and inaccessible even with the highest system privileges, side-channel analysis is gaining interest in mobile forensics.

When an IC chip operates on a circuit board, electromagnetic emanations are generated based on the current consumed by the internal transistors. By capturing these signals multiple times and correlating the collected data with a power consumption model of the transistors, secret information stored within the target IC may be revealed, allowing the attacker to access internal data [23]. This type of analysis, known as Side Channel Analysis (SCA), has been a prominent area of security research, particularly in the context of smart cards and other security technologies. Recent research has demonstrated that SCA can be used to retrieve cryptographic keys from the application processors in modern mobile devices [24, 25].

Although each application processor is unique and requires specific research, SCA is a promising technique for acquiring cryptographic keys from modern mobile devices. Once obtained, these keys can be used to decrypt data such as bootloaders. Additionally, by reverse engineering the key derivation process, an examiner may conduct offline brute-force attacks to identify the user passcode of the target device [25].

2.4 Mobile Forensics Through Non-Volatile Memory Storage

Given the challenges and techniques discussed above, current mobile forensic methods increasingly focus on reverse engineering the application processor within a SoC and the software it runs. Simple chip-off analysis is no longer sufficient for effective data recovery, which has led to a decline in forensic techniques that rely solely on memory chips. Meanwhile, non-volatile memory technology is becoming increasingly complex, with memory chips now containing multiple logic silicon dies within a single package. Consequently, it remains crucial to thoroughly investigate non-volatile memory chips used in mobile devices.

2.4.1 Hardware Reverse Engineering

NAND flash memory remains the most widely used technology in modern storage solutions due to its high density, cost-effectiveness, and non-volatile nature. However, the direct integration of raw NAND flash memory into digital devices is becoming less common as storage technologies evolve. Instead, NAND flash is incorporated into more sophisticated, multi-layered storage solutions such as embedded MultiMediaCard (eMMC), Universal Flash Storage (UFS), Non-volatile Memory express (NVMe), and UFS-based Multi Chip Package (uMCP). Apple products utilize proprietary controllers in their flash memory-based storage solutions, enhancing performance and security. These storage solutions encapsulate flash memory and the controller in a single package, adding complexity to forensic investigations. The controller manages tasks such as wear leveling, error correction, and encryption, effectively acting as a gatekeeper to the data stored within the flash memory.

Understanding the inner workings of these storage devices and extracting valuable information from the internal flash memory necessitates hardware-based reverse engineering. This involves identifying the required voltages, commands, and signals to access the memory, as well as overcoming any security measures implemented by the controller. By successfully interfacing with these storage devices, forensic investigators can potentially access hidden or protected data areas, which may contain critical information such as security keys or remnants of deleted files. Access to such hidden data is invaluable for forensic analysis, as it can provide insights that are otherwise inaccessible through standard data extraction methods.

Expanding forensic capabilities to include the reverse engineering of storage devices is becoming crucial, given the increasing complexity of modern mobile storage solutions. As manufacturers continue to enhance the security and performance of their products, the role of hardware-based analysis in forensic investigations will only grow in importance, offering the potential to unlock critical evidence hidden within non-volatile memory.

2.4.2 Physical Data Extraction for Deleted Data Recovery

Once the appropriate method for accessing stored data in flash memory is determined, forensic investigators can proceed with the extraction of data from all physical addresses of the target flash memory. This process involves reading data at the lowest level, bypassing the logical structures imposed by the OS and internal memory management. Advanced storage solutions, such as eMMC and UFS, contain internal controllers that manage the

translation of data read and write commands issued by the host system to the underlying flash memory. In scenarios such as factory resets or when a device undergoes data wiping, the erase commands from the host system is translated by the internal controller, rendering the data inaccessible under normal conditions. However, because the actual data blocks often remain intact, examining the entire physical memory space is essential for comprehensive data recovery in mobile forensics. A detailed analysis can reveal hidden or residual data that may include critical evidence, such as previously deleted files, metadata, or other forms of digital artifacts.

Forensic data recovery efforts must therefore focus on direct interaction with the memory chip at the physical level, utilising the knowledge obtained through hardware reverse-engineering. This approach allows investigators to retrieve data that is no longer accessible through the device's standard interfaces, thereby increasing the chances of recovering crucial information for forensic analysis.

2.4.3 Exploiting Security Features in Flash Memory

Aforementioned advanced storage solutions incorporate various security features, including passcode locks, secure erase functionalities, and Replay Protected Memory Block (RPMB), designed to protect critical data on the device from unauthorized access. These security mechanisms are managed by the internal controller embedded within the memory chip, which runs proprietary vendor-specific firmware.

Reverse-engineering the firmware running on the internal controller is one of the most effective methods for understanding and potentially circumventing these security features. Analyzing this firmware may reveal vulnerabilities that can be exploited to execute arbitrary code, thereby disabling the security mechanisms. However, in most cases, the firmware is inaccessible due to its proprietary nature, making memory storage devices a black box for forensic investigators. As a result, investigators often rely on a black-box approach, monitoring inputs and outputs to bypass security features indirectly. Despite these challenges, leveraging such techniques allows forensic experts to develop methods for overcoming built-in security features, enabling effective data extraction and analysis from devices utilizing advanced storage technologies.

2.5 Conclusion

Due to the growing concerns over security and privacy among mobile device users, driven by the valuable financial and personal data stored on these devices, manufacturers are in-

creasingly implementing robust encryption and other advanced security mechanisms. This trend has significantly impacted traditional forensic data acquisition methods, as these enhanced security measures are designed to protect sensitive user information from unauthorized access.

Historically, acquiring raw data from the non-volatile memory of a mobile device would often yield valuable information for criminal investigations. However, as discussed in this chapter, contemporary physical data acquisition techniques frequently fall short of producing human-readable data due to sophisticated encryption protocols. Moreover, additional security features, such as secure boot processes and access control mechanisms, have made it increasingly difficult for forensic examiners to extract even live data from target devices.

As a result, the ability to bypass or disable device locks and encryption, while maintaining the integrity of user data, has become a critical objective for forensic investigators working with modern mobile devices. To achieve this, extensive reverse engineering and the exploitation of security vulnerabilities are now essential components of forensic methodologies. While much of the vulnerability-based research has traditionally concentrated on the SoCs and security chips of target devices, the complex architecture and multiple layers of hardware present in modern memory chips offer a promising avenue for further exploration. Exploiting the security features of memory chips themselves could therefore represent a significant frontier in the evolving landscape of mobile forensics.

3 Effective Forensic Data Recovery from Flash Memory

Flash memory is the most commonly used storage technology in modern mobile devices due to its low cost-per-bit and high storage density. This chapter explores the intricacies of flash memory technology in detail and examines its impact on forensic data recovery methods.

3.1 Challenges in NAND Flash Memory Forensics and Mitigation Techniques for Data Recovery

NAND flash memory continues to increase in popularity as a storage medium for a wide range of devices, such as smartphones, thumb drives, and Solid State Drives (SSDs). As a result, digital forensic investigators have been encountering significantly more NAND flash memory based devices than before during the course of criminal investigations. When an operational NAND flash memory based device is received for analysis, investigators can use logical data extraction, where data can be read out using an interface provided by the device vendor [9]. Commercial software-based forensic acquisition tools automate logical data extraction, and can yield sufficient data from the device. Unfortunately, a device received as part of an investigation may be physically damaged, or the device may not provide an interface for data acquisition, and as a result, its data may be inaccessible using the automated software-based approach. In these cases, digital forensic investigators must

physically remove the NAND flash memory chip from the Printed Circuit Board (PCB) inside the device [26]. Once the chip has been removed, investigators can perform low-level analysis on the chip, at which point the data that was originally on the chip can potentially be recovered. This analysis method is commonly referred to as chip-off analysis.

Previous research on forensic low-level analysis of NAND flash memory chips has focused on reverse engineering the techniques implemented by the original NAND flash memory controllers, in order to access the data residing on the chip. Breeuwsma et al. established a thorough forensic data recovery procedure from a NAND flash memory chip, going from acquiring the physical image of the data on the NAND flash memory chip to reconstructing the file system used to store the data, by reverse engineering multiple techniques that are implemented in NAND flash memory controllers. For relatively old devices, where the number of raw bit errors that occurred within the device are low, this data recovery procedure is often sufficient. However, as NAND flash memory has scaled down aggressively in process technology node size to enable higher storage capacity, the number of raw bit errors that occur in NAND flash memory has increased by several orders of magnitude, as demonstrated experimentally in Cai et al. [27], Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai [28], Cai et al. [29]. As a result, the procedure proposed by Breeuwsma et al. [12] often cannot adequately recover the data in modern NAND flash memory.

In order to ensure that data retrieved from NAND flash memory by an end user does not contain any errors despite the increasing occurrence of raw bit errors, modern NAND flash memory controllers employ sophisticated Error Correction Codes (ECCs), such as BCH codes [30, 31, 32] or LDPC codes [33, 34]. These codes can correct up to a fixed number of raw bit errors for every read operation. Likewise, to maintain the integrity of digital evidence extracted from a device that uses NAND flash memory as its storage medium, forensic investigators need to correct errors that appear in the raw data extracted from the device. Thus, it is essential for the forensic data recovery procedure to extract the ECC information stored within the chip and use this information to correct the errors. In addition to ECC, many modern flash controllers employ data randomization techniques, where data that is written into memory is scrambled by XORing the data with a reproducible pseudo-random number, to reduce the impact of data value dependence on reliability [35, 36]. Zandwijk [37] provides a mathematical approach to reverse engineer both the ECC and data randomization algorithms from the raw data that is extracted from the NAND flash memory chip.

However, even when the ECC and data randomization algorithms are correctly identified and used to decode the raw data extracted from the NAND flash memory chip, digital

forensic examiners may find that many chunks of data contain more errors than the ECC algorithm is able to correct. The ECC codeword contains only enough information to correct up to e bits within an n-bit data chunk (hereafter e represents the error correction capability). If the data chunk contains more than e errors, the errors are *uncorrectable*, and the data cannot be successfully recovered. This compromises the integrity of the recovered data for forensic analysis.

In order to perform chip-off analysis, forensic investigators follow best practices used by electronics manufacturers for their *rework* process, where manufacturers remove and replace faulty components in their products ([38, 39]). This procedure uses hot air to heat the chip just enough to melt the solder that connects the chip to the PCB, which allows the safe removal of the chip. This technique is referred to as thermal-based chip removal. Even though the temperature used during chip removal is the minimal temperature necessary for the solder to reach its melting point (usually more than 200°C), this temperature is still high enough to introduce a very large number of new raw bit errors into the chip. Unfortunately, because the ECC data stored on-chip has a limited error correction capability, newly introduced errors can often make a significant portion of the data on a NAND flash memory chip unrecoverable with traditional low-level analysis techniques.

To mitigate the impact of the new errors introduced during chip removal, the dynamic mechanisms implemented within the NAND flash memory are examined. In particular, the *read-retry* mechanism [28] will be studied in this chapter. In a NAND flash memory cell, data is stored in the form of the threshold voltage of the cell's floating-gate transistor (i.e., the voltage at which a transistor turns on). In order to read data from the NAND flash memory cell, a read reference voltage is applied to the transistor. If the read reference voltage is higher than the threshold voltage, the cell turns on; otherwise, the cell turns off. As several prior works have shown [27, 28, 29, 40, 41, 42], the threshold voltage of a floating-gate transistor can shift over time, due to continuous leakage of charge from the transistor. Since the standard read operation uses the default read reference voltage, it is unable to account for such a threshold voltage shift, and thus the read operation introduces an error (e.g., a bit value 1 was read even though the stored value was 0). The flash controller can mitigate these errors by dynamically adjusting the read reference voltage to compensate for the threshold voltage shift. This mechanism is known as *read-retry* [28, 40]. Modern NAND flash memory chips include several variants of read-retry, which use different techniques to adjust the read reference voltage.

3

3.2 Multi-Level Cell NAND Flash Memory Basics

In this section, the design and operation of planer multi-level cell (MLC) NAND flash memory, one of the basic and common NAND flash memory technologies, are explained. Readers are also referred to previous studies for detailed information on the operation of flash memory in [27, 29, 43, 44, 41, 45, 46].

3.2.1 Flash Memory Organization

NAND flash memory stores data within an array of flash cells. A cell consists of a single floating-gate transistor, where the floating gate of the transistor can store some amount of charge, as shown in Figure 3.1. The charge stored (i.e., trapped) within the floating gate determines the threshold voltage at which the transistor turns on. Oxide layers are placed above and below the floating gate to prevent the stored charge from leaking out of the floating gate. To program a flash cell to a specific threshold voltage, a high voltage is applied to the transistor's control gate.

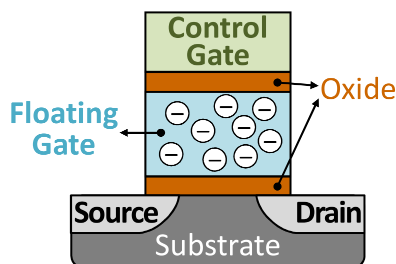


Figure 3.1: A flash memory cell, which consists of a floating-gate transistor.

The threshold voltage can be programmed to a voltage level within a fixed range. This fixed range is split up into multiple voltage windows, or states, where each state represents a certain bit value. If the each flash cell stores a one-bit value (i.e., 0 or 1), it is called single-level cell (SLC) flash memory. In order to provide higher storage density, NAND flash memory manufacturers developed Multi Level Cell (MLC) technology. An MLC stores more than two bits of data within a single cell. If a cell stores 2 bits of data, the voltage range is split into four states (*ER*, *P*1, *P*2, and *P*3), with each state corresponding to one of the data values 0b00, 0b01, 0b10, or 0b11, as shown in Figure 3.2. The number of states can be further increased, such as eight states for a cell that stores three bits of data. In this thesis, the term MLC specifically refers to NAND flash that stores 2 bits per cell, even though MLC can technically include flash cells storing more than two bits. This focus allows for

clearer explanations and more straightforward analysis throughout the work.

Due to variation during programming, the threshold voltage of cells programmed to the same state is distributed across the voltage window for the state. This results in a threshold voltage distribution of flash cells across the voltage range, as shown in Figure 3.2.

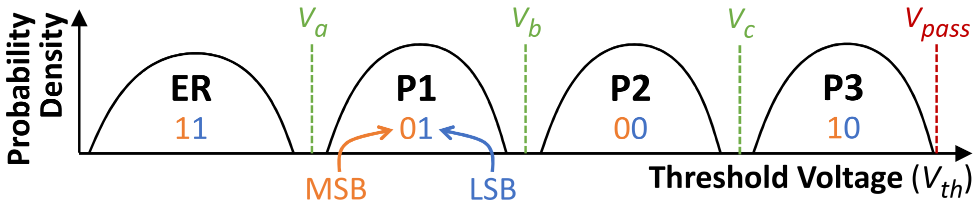


Figure 3.2: Threshold voltage distribution of cells in MLC NAND flash.

A NAND flash memory chip contains thousands of *flash blocks*, which are two-dimensional arrays of flash cells. Figure 3.3 shows the internal organization of a block. Each block contains dozens of rows (i.e., *wordlines*) of flash cells. All of the cells on the same wordline are read and programmed as a single group. MLC NAND flash memory partitions the two bits of data in each cell across two separate *pages* (the unit of data programmed at a time). As Figure 3.2 shows, the two bits stored within an MLC are referred to as the Least Significant Bit (LSB) and the Most Significant Bit (MSB). The LSBs of all cells on one wordline form the LSB page of that wordline (e.g., Page 1 of Wordline 1 in Figure 3.3), and the MSBs of these cells form the MSB page (e.g., Page 4 of Wordline 1). For 2y-nm (i.e., 20–24nm) NAND flash memory, a single page consists of between 4–16KB of data [47]. Within each column of flash cells in the block, the sources and drains of the cells' transistors are connected in series to form a *bitline*. The cells on a bitline share a common ground on one end, and are connected to a sense amplifier on the other end.

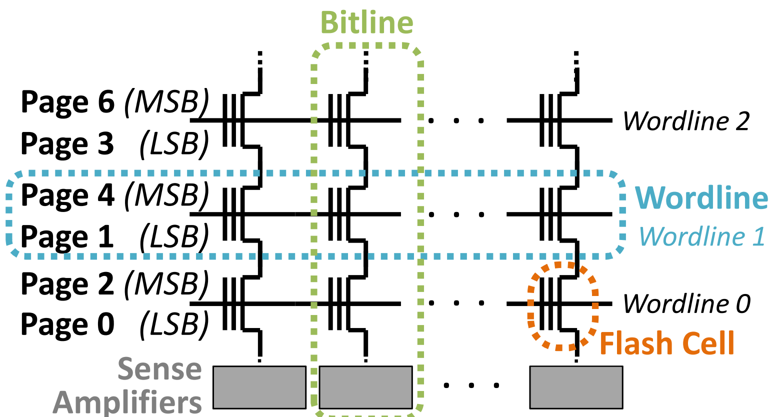


Figure 3.3: Organization of a NAND flash block.

3.2.2 Programming and Erasing Data

To program data into a flash cell, the cell needs to be in the erased state (i.e., no charge should be stored within the floating gate of the cell). During a program operation, electrons are injected into the floating gate by applying a high positive voltage to the control gate (see Figure 3.1 for a diagram of the flash cell). NAND flash memory uses a procedure known as Incremental Step Pulse Programming (ISPP) [48]. During ISPP, the high programming voltage is applied for a very short period, known as a step-pulse. ISPP then checks the current voltage of the cell. ISPP repeats the process of applying a step-pulse and checking the voltage until the cell reaches its target threshold voltage. If the data that currently exists in a cell needs to be overwritten, the data in the cell first needs to be erased. Within NAND flash memory, an erase operation is performed at block granularity (i.e., an entire block of flash cells is erased at once).

Over time, as a cell is programmed and erased, the cell begins to wear out, reducing its ability to reliably store charge within the floating gate [49, 50]. As this wearout is a result of the number of times a cell is programmed and erased, the degree of wearout is quantified in terms of program/erase (P/E) cycles, as done in many prior works [27, 29, 43, 44, 28, 51, 52, 53, 40].

3.2.3 Reading Data from NAND Flash

To read a page of data from a block, the flash controller applies a *read reference voltage* to the cell's control gate. If the threshold voltage of a cell is lower than the read reference voltage, the cell switches *on*; otherwise, the cell switches *off*. The read reference voltage used to read a cell depends on which page is being read from the wordline. As shown in Figure 3.2, to determine the LSB of a cell, the controller applies a single read reference voltage, V_b. If the threshold voltage of the cell is lower than V_b, the cell is in either the ER state or the P1 state, and holds an LSB of 1; otherwise, the cell is either in the P2 state or the P3 state, and holds an LSB of 0. To determine the MSB of a cell, the controller applies *two* read reference voltages, V_a and V_c. The two voltages allow the controller to determine if a cell is in the P1/P2 states, and holds an MSB of 0, or if the cell is in the ER/P3 states, and holds an MSB of 1.

Since multiple cells are tied together on a single bitline, it is necessary to ensure that the cells not being read pass through the data being output from the cell intended for reading. To achieve this, the flash controller applies a *pass-through voltage* to the control gate of each unread cell (V_{pass} in Figure 3.2). The pass-through voltage is higher than any threshold

voltage that can be stored within a flash cell, ensuring that a cell *not* being read is always turned *on* during the read operation, allowing the data from the target cell to successfully reach the sense amplifier.

3.2.4 Correcting Errors

Data stored within flash cells can often contain errors. An error is introduced when the threshold voltage of a cell shifts outside of the voltage window to which the cell was originally programmed. There are a number of sources of errors within flash memory [27, 29], such as retention loss [51, 40], cell wearout [27, 28, 54], cell-to-cell program interference [52, 53], and read disturb [43]. As flash cells scale down to smaller process technology nodes, the total amount of charge that each cell can store decreases, which, in turn, increases the susceptibility of the flash cells to errors [55].

In order to combat the errors contained within the cells (which is referred to as *raw bit errors*), NAND flash memory makes use of Error Correction Codes (ECCs) [27, 56]. When data is programmed to a flash page, an ECC codeword is also written, which contains enough redundancy to correct e bits out of the n-bit data [37]. e is referred to as the *error correction capability* of the codeword. When the flash page is subsequently read, the ECC codeword is sent alongside the data to the flash controller. Inside the controller, both the data and the ECC codeword are input to the ECC logic, which checks for errors using the implemented error correction algorithm. Based on the results of the algorithm, the controller fixes the erroneous bits in the data, if any, and returns the corrected data value. If the data read from the page contains no more than e raw bit errors, the controller successfully returns correct data to the end user. If the data read from the page contains more than e raw bit errors, full data correction is not possible, and the page data is said to be *corrupted* (i.e., it is uncorrectable) [57, 58]. A block that contains corrupted data is marked by the flash controller as a *bad block* [59], and is no longer used for storing data.

3.3 Bit Errors in Flash Memory During Forensic Analysis

Bit errors in flash memory can occur when the threshold voltage of a flash cell shifts. The probability of occurrence of such shifts has increased due to continued device scaling, which allows manufacturers to increase the flash storage density [55]. The reliability of NAND flash memory has been widely researched For a detailed overview of MLC NAND reliability, we refer the reader to Zambelli et al. [60] and Cai et al. [29]. Cai et al. investigated multiple error factors on MLC NAND flash memory, including program/erase errors, program

interference errors, retention errors, and read errors. Digital forensic investigators should assume that the majority of NAND flash memory devices they receive for analysis already contain multiple errors at the time the device is obtained.

In this section, the two major sources of errors are investigated. Namely, the one that can be introduced during digital forensic analysis, in addition to the preexisting errors within the NAND flash memory device. First, a device is often stored unused for several days or even weeks before investigators are able to examine its contents, due to issues such as transport time or lab backlog [61]. During this time, additional *retention errors* can occur, which is discussed in Section 3.3.1. Second, for devices where thermal-based chip removal is required, a high temperature must be steadily applied to the device. This high temperature rapidly accelerates the effect of retention errors, as shown in Section 3.3.2.

3.3.1 Retention Errors

Retention errors in a flash cell occur when the charge stored within the floating gate of cell transistor *leaks*. Due to the structure of the cell transistor (see Figure 3.1), where the floating gate and substrate are separated by an oxide layer, a small amount of charge *tunnels* through the oxide, causing the leakage. This trend accelerates as the P/E cycle count increases [27, 55], as repeated programming and erasing of a cell degrades the oxide layer, which in turn allows charge to tunnel through the oxide layer at an increasing rate [40]. This tunneling occurs whether or not a NAND flash memory device is powered up, causing retention errors to accumulate when the device is stored unused with or without power.

Imagine a hypothetical case where a device was seized at a crime scene, and the device is received for analysis at a digital forensics lab three weeks later. Let us assume that this device had been used over the course of three years, and that the NAND flash memory within the device endured 10 P/E cycles each day during these three years of usage, adding up to a total of approximately 10^4 P/E cycles over its lifetime. Figure 3.4 shows the relationship between the P/E cycle count and the retention error rate found by Cai et al. [27] for 3x-nm MLC NAND flash memory chips. The figure demonstrates that the error rate grows with (1) the P/E cycle count and (2) the *retention age* (i.e., the time elapsed since the data was programmed). By comparing the curve labeled *1 Day* with the curve labeled *3 Weeks*, it is evident that the number of errors in the device increases by 38 times over the three-week period between device seizure and lab delivery, while the device is being stored and transported.

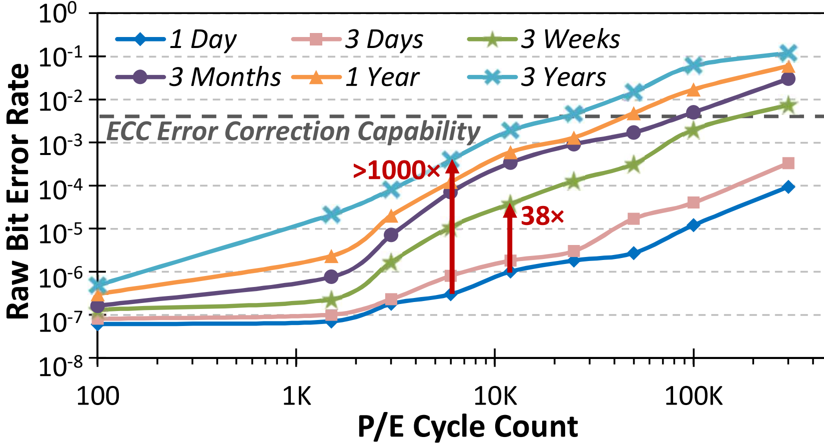


Figure 3.4: Raw bit error rate of a 3x-nm NAND flash memory chip for different retention ages vs. P/E cycle count. Reproduced from [27].

3.3.2 Thermal Effect on Error Rate

After the device is received by the digital forensics lab, investigators must extract the data from the device. In many such cases, the device might have been damaged prior to seizure, and cannot be accessed using software-based analysis techniques. In these cases, a chip-off analysis (Section 2.2.1) must be performed, where investigators physically remove the NAND flash memory chips from the device and use hardware that can extract data directly from the chips [26]. In order to remove the chips, investigators try to melt the solder connecting the chips to the PCB, at which point the chips can be pulled off.

Unfortunately, this thermal-based chip removal procedure greatly accelerates the number of retention errors that occur. Mielke et al. [62] states that when NAND flash memory is exposed to high temperature, Arrhenius' Law [63, 64] can be used to convert the effects of high temperature into additional data retention time at normal operating temperature for the memory. Let t_b denote the amount of time that heat is applied to the chip, and t_r denote the equivalent retention age at the normal operating temperature. According to Arrhenius' Law, t_r can be calculated as:

$$t_r = t_b \cdot exp[\frac{E_a}{k}(\frac{1}{T_r} - \frac{1}{T_b})] \tag{3.1}$$

where k is the Boltzmann constant, T_r denotes the normal operation temperature, and T_b denotes the *baking temperature* (e.g., the high temperature applied to the chip during the removal process). E_a is the activation energy, set to 1.1 eV according to Mielke et al. [65].

Following standard rework procedures used by electronics manufacturers [38], thermal-based chip removal requires investigators to apply 250°C of heat for a duration of approximately two minutes. Using Equation 3.1, it is evident that applying this heat introduces the same number of retention errors as leaving the NAND flash memory untouched at room temperature for *833 years*. Given that the amount of retention errors after *only three years*, as shown in Figure 3.4, already introduces more than 1000 times the number of errors into NAND flash memory, compared to the number of retention errors after one day, it can be extrapolated that 833 years' worth of retention errors would be significantly and prohibitively larger (over 10^5 times the number of retention errors after one day). Such a large rate of errors could easily overwhelm the error correction capability of ECC algorithms employed in modern flash controllers.

3.4 Read Retry Features in Flash Memory

As discussed in Section 3.3, the number of errors that exist in the raw data can increase by several orders of magnitude even under best practices used in forensic investigation techniques. If left unaddressed, the number of errors can quickly exceed the total error correction capability of the flash device, which in turn results in partial or complete data loss during forensic data recovery. In this section, a mechanism that exists in modern MLC NAND flash memory chips, called *read-retry*, is examined. It can be used as part of the recovery process to mitigate the high number of errors introduced during data recovery.

Recall from Section 3.2 that errors occur when the threshold voltage of a flash cell shifts, causing the read reference voltages to incorrectly interpret the state of the cell. Prior work has demonstrated that retention errors are the dominant source of errors in MLC NAND flash memory [27]. As a result, the threshold voltage of a flash cell tends to decrease as the data retention age increases. To combat this decrease in threshold voltage, NAND flash memory manufacturers provide a read-retry operation, which can adjust the read reference voltages used to read data from a cell, and thus potentially reduce the number of raw bit errors in the data [28].

Figure 3.5 shows an example of a threshold voltage distribution that has shifted downwards due to charge leakage over retention time. For the sake of simplicity, only the ER state and P1 state distributions are shown in the figure. If the normal read reference voltages (i.e., V_a, V_b, and V_c in Figure 3.2) are applied to the original distribution (i.e., before the distribution shifted), the values of all cells in the distribution can be read out correctly. Once the distribution shifts, the read reference voltages no longer fall in between the shifted distri-

butions of each voltage state, but instead fall inside the distributions of some of the states. For example, the default read reference voltage V_a is used to distinguish between cells in the ER state and those in the P1 state. As shown in Figure 3.5, after the distribution shifts, some cells in the P1 state now fall to the left of V_a. If the controller continues to use V_a, it incorrectly classifies these cells as being in the ER state. As a result, the default read reference voltages (V_a, V_b, V_c) can introduce many raw bit errors when the threshold voltage distribution shifts.

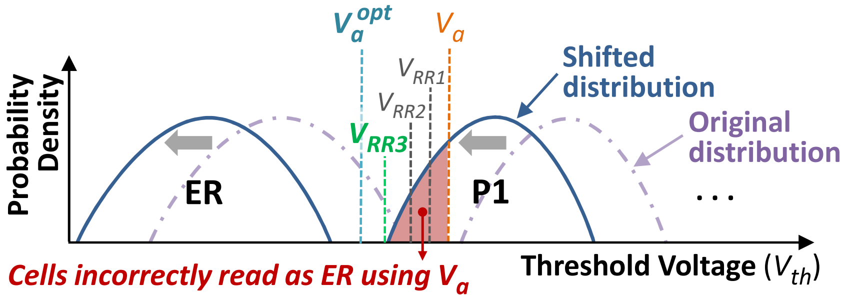


Figure 3.5: Effect of read-retry operation on a shifted threshold voltage distribution (showing the distributions for only the ER state and the P1 state).

The read-retry operation can be applied to the example shifted distribution to compensate for the effects of the voltage distribution shifts caused by charge leakage. The basic goal of the read-retry operation is to adjust the read reference voltages up or down with the goal of minimizing the errors that are introduced due to misclassification. The optimal read reference voltages (i.e., the voltages that are exactly in the middle of the distance between two neighboring distributions, which minimizes the number of errors) are referred as V_a^{opt}, V_b^{opt}, and V_c^{opt}. One example read-retry mechanism can adjust the voltages down one step at a time during a read operation, checking to see whether the number of errors goes down with each subsequent step. Figure 3.5 illustrates how this example mechanism works. Here, the read-retry mechanism tries to adjust the voltage used to distinguish between cells in the ER state and cells in the P1 state. The mechanism tries several voltages (V_{RRn} in the figure, where n represents the nth voltage tried). As shown in Figure 3.5, the mechanism eventually finds a voltage (V_{RR3}) where there are no cell classification errors (because V_{RR3} falls *between* the threshold voltage distributions of the ER and P1 states). Even though V_{RR3} is higher than V_a^{opt}, V_{RR3} can still be used to safely extract data from the NAND flash memory *without any errors*.

Note that the read-retry operation itself does not always reduce errors. Furthermore, the details of read-retry modes are often proprietary and not always publicly available, mak-

ing it difficult to learn whether a mode supports the ability to ensure that the number of read errors never increases. While retention errors usually shift the threshold voltage of a cell down, the threshold voltage of a cell can sometimes increase from its original state, as a result of cell-to-cell interference that occurs during programming, reading, and erasing [52, 53, 43]. The direction and magnitude of the change in threshold voltage vary for each cell, due to factors such as the retention age of the data in the cell, the number of read and program operations performed to neighboring cells, and manufacturing process variation. As a result, the threshold voltage distributions of each state do not shift uniformly, and can overlap with each other. Therefore, it is possible that simply shifting the read reference voltage up or down (without checking the resulting number of errors) could unintentionally introduce more errors than the normal read that is done with the default read reference voltage. Therefore, read-retry based mechanism needs to be used with an error-correction mechanism in order to make sure that the applied voltage is appropriate to reduce the bit errors.

3.5 Experimental Evaluation of Enhancing Reliability in Chip-off Analysis through Read Retry

This section evaluates the reliability impact of chip-off digital forensic analysis on MLC NAND flash memory chips. The effects of data retention on the Raw Bit Error Rate (RBER) of two MLC NAND flash memory chips are examined. Subsequently, the impact of applying high temperatures—simulating conditions used in thermal-based chip removal during chip-off analysis—on the RBER is assessed. Finally, the changes in error rates with the use of the read-retry mechanism are demonstrated.

3.5.1 Testing Methodology

To investigate the raw bit errors encountered during chip-off analysis (see Section 3.3.2) and to evaluate the effectiveness of the read-retry operation (see Section 3.4) in mitigating these errors, the effects of data retention and thermal-based chip removal were examined using two new 2y-nm NAND flash memory chips from different manufacturers. These chips are referred to as Chip A and Chip B. An Altera DE0 Field Programmable Gate Array (FPGA) board [66] was used to design a controller that interfaces with the target chips, following methods similar to those described by Cai et al. [67] and Breeuwsma et al. [12]. Figure 3.6 illustrates the testing environment. The FPGA issues commands to and receives data from the NAND flash memory chips. A USB microcontroller transfers the

data from the NAND flash memory chips to the PC, where it is collected and analyzed. All testing was conducted at room temperature, unless otherwise specified.

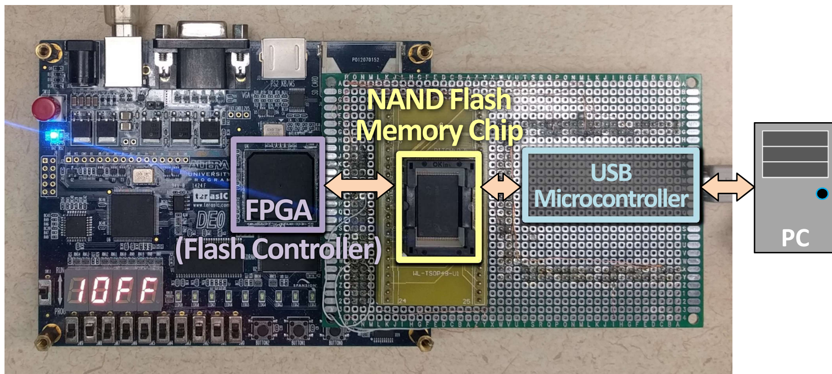


Figure 3.6: Photograph of our test infrastructure used to extract data from MLC NAND flash memory chips.

The retention errors encountered during forensic examinations are first investigated. Multiple blocks are randomly selected from the target chips, into which pseudo-random data is programmed. This approach simulates the data scrambling used by modern flash controllers [35, 36]. To assess the impact of retention errors on both relatively new and heavily-used devices, the blocks are divided into subsets, with each subset undergoing a distinct number of Program/Erase (P/E) operations. The P/E cycle counts chosen for this study are 10, 300, 1000, 2500, and 4000, covering a broad range of device wear. After reaching the target P/E cycle count, reads are performed at specified retention ages to analyze the effect of retention age on the error rate. The retention ages studied include Day 0 (i.e., immediately after programming), Day 1, and Weeks 1, 2, 3, and 4.

The thermal-based chip removal procedure is then simulated by baking the chips after the stored data reaches a specified retention age. The chips are exposed to a temperature of 250°C for two minutes using a heat gun, replicating the conditions used during chip removal in chip-off analysis. Following the baking procedure, data read operations are immediately conducted to measure the resulting error rate. The effects of this thermal exposure are examined at three different retention ages: Day 0 (i.e., immediately after programming), Week 1, and Week 4.

One of the target chips (Chip B) has the read-retry operation implemented. For this chip, multiple read operations were performed at every retention age, performing one read for each available read-retry mode, and one read without read-retry. The read-retry modes in Chip B does not have the ability to check or observe whether the number of errors is

lower than that when the default read reference voltages are used. Each mode simply shifts the read reference voltage to a different fixed level.

3.5.2 Errors Introduced Due to Retention Time

Figure 3.7 shows how the RBER of the tested NAND flash memory chips varies with (1) the P/E cycle count of the chip (i.e., how much the flash memory cells on the chip have been worn out), and (2) the retention age of the data (i.e., the amount of time that elapses after the data was written).

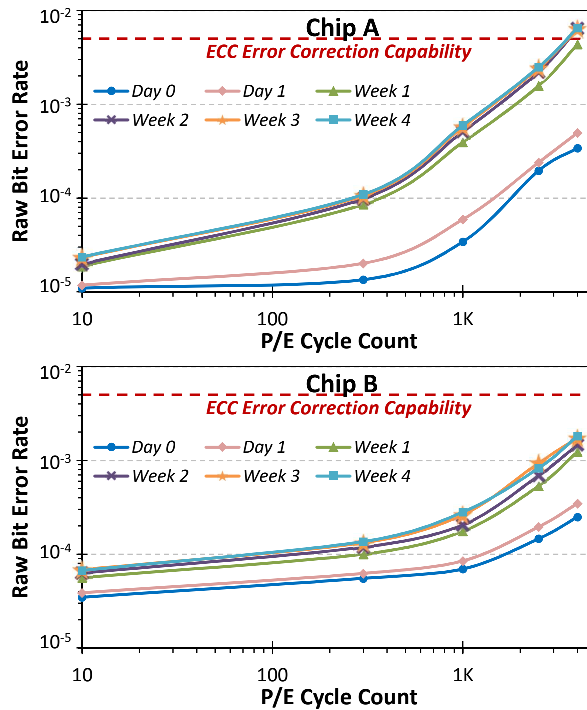


Figure 3.7: Raw bit error rate at different data retention ages, for different P/E cycle counts.

In both chips, the RBER grows as the retention age increases. The Day 0 results in Figure 3.7 show the number of errors that exist in flash blocks immediately after the data is programmed to the NAND flash chips. The RBER increases over time, but the largest increases occur soon after the the data is programmed. For example, the increase in RBER is much greater during the first week (6.34 times for Chip A and 1.81 times for Chip B at 300 P/E cycles, compared to Day 0 of each respective chip) than the second week (1.15 times for Chip A and 1.19 times for Chip B), and greater during the second week than the third week (1.08 times for Chip A and 1.12 times for Chip B).

Additionally, in both chips, the RBER grows as the P/E cycle count increases. In other words, as a NAND flash memory chip is worn out, its susceptibility to raw bit errors due to retention increases, for data with the same retention age. Note that the y-axis in Figure 3.7 is in log scale. A chip at a higher P/E cycle count (i.e., a chip with greater wearout) accumulates retention errors at a much faster rate than a chip at a lower P/E cycle count [40].

Furthermore, it was observed that while the RBER grows with both wearout and retention age, the overall RBER of the chip does not exceed the error correction capability of ECC unless the P/E cycle count significantly exceeds the endurance guaranteed by manufacturers. For 2y-nm MLC NAND flash memory, a controller that employs BCH codewords [30, 31, 32] for ECC can typically correct 40 bits of errors for every 1KB of data [68] (i.e., it can correct errors for a RBER of up to 4.9×10^{-3}). As the results from Figure 3.7 show, the overall RBER stays lower than this error correction capability through a retention age of four weeks, for P/E cycle counts below 3000 cycles, which is the typical endurance of commercial 2y-nm MLC NAND flash memory [68].

Within the experimental platform, a 70-byte BCH codeword is implemented for ECC to provide the expected 40 bits of correction capability for each 1KB chunk of data. Note that a data chunk is smaller than a page. When read operation is performed, the BCH codeword is used to determine how many of the data chunks cannot be successfully corrected by ECC.

Table 3.1 shows the fraction of pages that contain at least one uncorrectable data chunk. Even before chip-off analysis is performed, if a chip has been worn out significantly (e.g., after 2500 P/E cycles for Chip A), it can contain some uncorrectable pages even at a retention age of just one week. However, for less worn-out chips (e.g., a chip at 1000 P/E cycles), none of the pages are uncorrectable even after a retention age of four weeks.

For perspective, a device can reach 2,190 P/E cycles if all of the pages in the NAND flash memory chip are written to twice a day, every day, over a period of three years. It is observed that once the chip exceeds the expected endurance of 2y-nm MLC NAND flash memory (3,000 P/E cycles), the fraction of pages with uncorrectable errors grows rapidly for Chip A. At 4000 P/E cycles, 64.9% of the pages in Chip A contain uncorrectable errors after a retention age of only one week.

Even when the overall RBER stays lower than the ECC error correction capability, errors in some pages become uncorrectable over time. For example, Chip A's RBER at a retention age of one week (i.e., seven days) at 2500 P/E cycles is 1.6×10^{-3} (see Figure 3.7), which is lower than the ECC error correction capability of 4.9×10^{-3}. At the same time,

Table 3.1: Fraction of pages containing uncorrectable 1KB data chunks. A dash (—) indicates that no data chunks are uncorrectable.

Chip	Retention Age (days)	P/E Cycle Count 10	300	1000	2500	4000
A	0	—	—	—	—	—
	1	—	—	—	—	—
	7	—	—	—	0.9%	64.9%
	14	—	—	—	9.6%	92.2%
	21	—	—	—	14.4%	91.8%
	28	—	—	—	15.9%	91.9%
B	0	—	—	—	—	—
	1	—	—	—	—	—
	7	—	—	—	—	—
	14	—	—	—	—	—
	21	—	—	—	—	0.0039%
	28	—	—	—	—	0.0065%

0.9% of the pages in Chip A contain uncorrectable errors, as shown in Table 3.1. Similar behavior is observed for Chip B, even though its overall RBER always remains below the error correction capability.

It is thus concluded that, even if all of the data inside the device is refreshed immediately before the device was confiscated, a worn-out device can quickly accumulate errors, and some of those errors become uncorrectable over time. Therefore, in order to avoid data loss due to uncorrectable data pages, data needs to be extracted from a NAND flash memory based device at the earliest possible time after the receipt of the device.

By investigating the state of each cell at various retention ages, a number of trends in the threshold voltage distribution shift can be characterized. A cell is defined as belonging to the set $[S_O, S_M]$ if it was originally programmed to state S_O but is misread as state S_M when using the default read reference voltages. The graphs in Figure 3.8 show the fraction of cells in Chip A that are in the set $[S_O, S_M]$, for all neighboring (S_O, S_M) pairs, out of the total number of cells originally programmed to state S_O, across a range of retention ages.

From Figure 3.8, it is observed that when S_M is a *lower* voltage state than S_O, a greater number of cells belong to $[S_O, S_M]$ as the retention age increases. For example, after a retention age of four weeks (i.e., 28 days), 0.20% of cells that were originally programmed to the P2 state are misread as belonging to the P1 state (i.e., the cells are in the set [P2, P1]), as opposed to only 0.02% after a retention age of one day. Regardless of the retention age, when S_M is a higher voltage state than S_O, only a very small number of cells belong to the

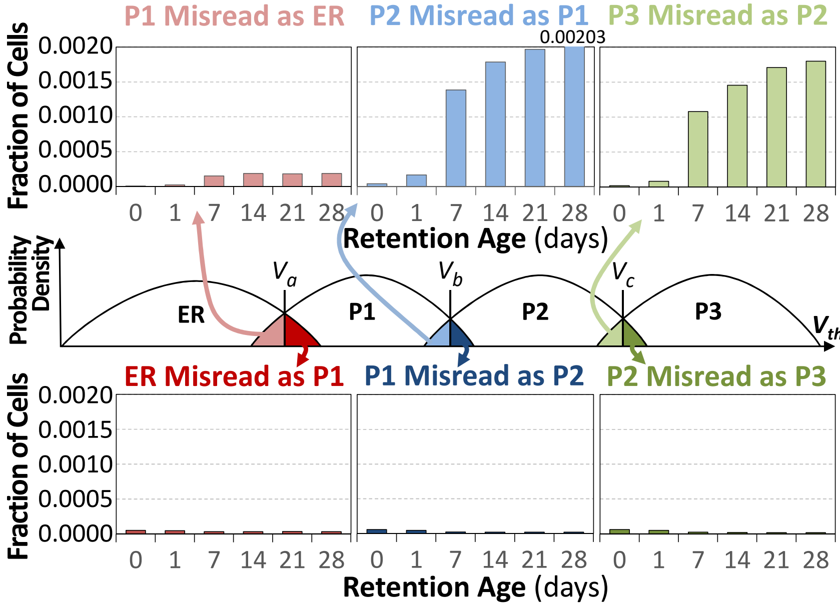


Figure 3.8: Fraction of cells in Chip A that were programmed to state S_O but are misread as belonging to state S_M, out of the total number of cells originally programmed to state S_O.

set $[S_O, S_M]$ (e.g., [ER, P1]).

From these results, it is found that the threshold voltage of a misread cell tends to be lower as the retention age increases. It can be concluded that the threshold voltage reduction, which occurs as a result of charge leakage from the floating gate of a flash memory cell, is the dominant source of errors that are introduced by retention age.

3.5.3 Errors Due to Thermal-Based Chip Removal

The study now focuses on how the RBER caused by retention errors changes following the thermal-based chip removal procedure used in chip-off analysis. Figure 3.9 shows the RBER after the chip baking process is performed to emulate the removal procedure. Note that the RBER data for Day 0 (i.e., immediately after programming), Week 1, and Week 4 before baking is the same as the data shown in Figure 3.7.

It is observed that simply applying the heat required for chip removal causes the RBER to increase significantly. When the heat is applied immediately after the data is written (*Day 0, After Baking* in the figure), at 1000 P/E cycles, the RBER increases by 432× for Chip A, and by 17× for Chip B, compared to the RBER before baking (*Day 0, Before Baking*). When the heat is applied four weeks after the data is written at the same P/E cycle count, (*Week 4, After Baking*), the RBER increases by 47 times and 54 times for

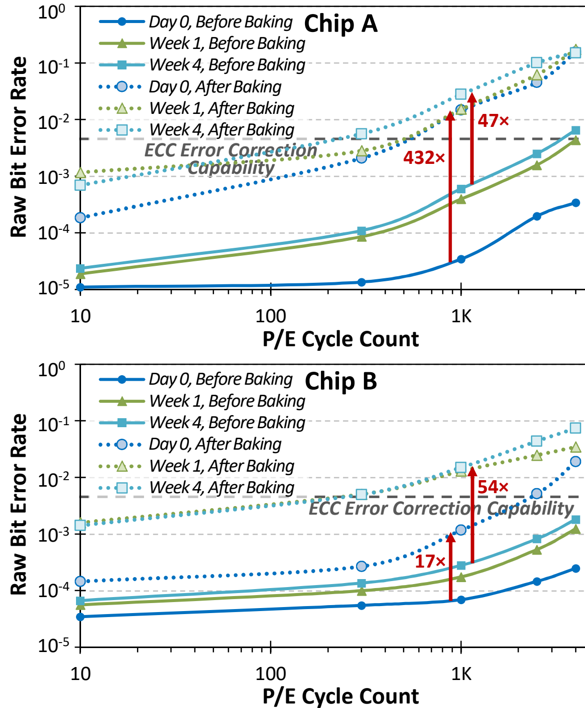


Figure 3.9: Raw bit error rate before and after baking NAND flash memory chips.

Chips A and B, respectively, compared to the RBER before baking (*Week 4, Before Baking*). Starting at 300 P/E cycles, the RBER exceeds the ECC error correction capability when heat is applied to the chip.

The impact of the heat applied during chip removal can cause critical damage to the data stored within the NAND flash memory chip that is being analyzed. Suppose that a digital forensic investigator starts the chip-off procedure four weeks after a device has been seized, and that the NAND flash memory chip inside the device was only lightly used (e.g., 300 P/E cycles) prior to seizure. Before applying heat, the RBER remains safely within the error correction capability of contemporary ECC, as shown in Figure 3.9 (*Week 4, Before Baking* at 300 P/E cycles), with a raw bit error rate of 1.1×10^{-4} for Chip A and 1.4×10^{-4} for Chip B. However, after applying the chip removal temperature, the RBER exceeds the error correction capability of ECC (*Week 4, After Baking* at 300 P/E cycles). The RBER becomes 5.6×10^{-3} and 5.0×10^{-3} for Chip A and Chip B, respectively. At such a high RBER, it is impossible to correct all of the errors in the data with the given ECC error correction capability. Thus, the integrity of the data recovery is compromised. As a point of comparison, The increase in RBER between Week 1 and Week 4 before baking for

both chips is extrapolated, to see how long it would take for the chips to reach the RBER after baking if we had not baked the chips. Baking can increase the RBER by 113 times for Chip A, and by 38 times for Chip B on average, while the increase in RBER between *Week 1, Before Baking* and *Week 4, Before Baking* is 1.43 times for both chips. Therefore, applying heat to a chip induces approximately two to five years' worth of retention errors at room temperature.

Table 3.2 shows the percentage of uncorrectable data chunks *after* the chips are baked (compare this to Table 3.1, which shows the uncorrectable data chunks *before* baking). Nearly all of the pages stored within the NAND flash memory contain uncorrectable errors after the baking process. At only 300 P/E cycles, 84.2% of the pages in Chip A and 83.6% of the pages in Chip B contain uncorrectable errors, and *all* pages contain errors when we reach 2500 P/E cycles for both Chip A and Chip B, when the heat is applied four weeks after the data is written. Based on the analysis, it is concluded that the thermal-based chip removal procedure, when left unmitigated, is prohibitively destructive, as it significantly reduces the amount of data that can be successfully retrieved from NAND flash memory during forensic recovery.

Table 3.2: Fraction of pages containing uncorrectable 1KB data chunks after applying heat to the chips. A dash (—) indicates that no data chunks are uncorrectable.

Chip	**Retention Age** (days)	**P/E Cycle Count**				
		10	**300**	**1000**	**2500**	**4000**
A	7	—	29.1%	99.8%	100.0%	100.0%
	28	0.7%	84.2%	96.9%	100.0%	100.00%
B	7	—	78.1%	96.5%	96.9%	96.9%
	28	—	83.6%	99.7%	100.0%	100.0%

3.5.4 Read-Retry Operation

The focus now shifts to evaluating the read-retry operation (see Section 3.4) and its effectiveness in addressing errors caused by the thermal-based chip removal process. This analysis employs the read-retry mechanism available in Chip B, which features two modes: Mode A and Mode B. *

Figure 3.10 illustrates the impact of the read-retry mechanism on the RBER as retention age increases, excluding the effects of the chip removal process. Across all P/E cycle counts, while the RBER rises with retention age when using the default read operation,

*No documentation is available from the manufacturer on how the modes operate.

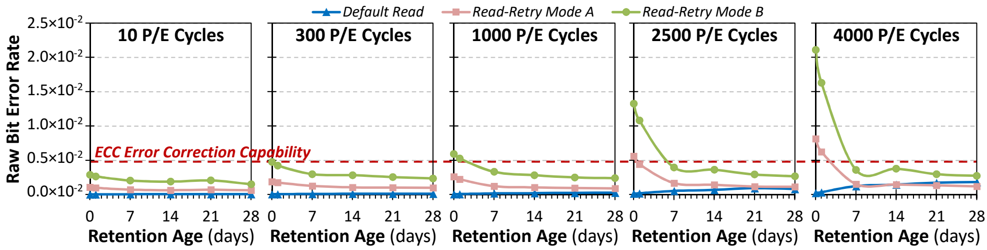


Figure 3.10: Effect of read-retry modes on the RBER of Chip B as retention age increases.

it decreases with retention age when either of the read-retry modes is employed. However, the read-retry modes appear to be relatively basic: Mode A outperforms the default read operation only at high P/E cycle counts. Under current conditions, where the RBER through normal reading is still within the ECC error correction capability, varying the read voltage via read-retry tends to increase the RBER. This is because the adjusted read voltage is not optimal for the cells' status in the target chip. Consequently, forensic investigators may not benefit from the read-retry operation in this scenario. Only when the chip is worn out beyond the manufacturer's endurance specification (e.g., at 4000 P/E cycles), Mode A effectively reduces the RBER compared to the default read operation after a retention age of two weeks (i.e., 14 days). Notably, the uncorrectable pages identified in Section 3.5.2 become correctable with the use of Mode A.

In contrast, when the baking process is applied to simulate thermal-based chip removal, the read-retry modes, despite their somewhat unpredictable behavior, prove to be effective in reducing the the RBER, as shown in Figure 3.11.

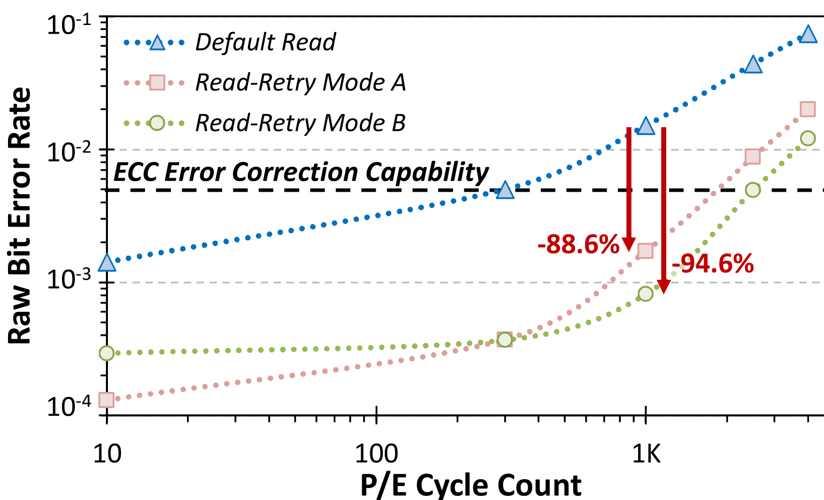


Figure 3.11: RBER with read-retry after Chip B is baked.

As detailed in Section 3.5.3, the baking process induces retention errors equivalent to 2–to 5 years' worth of degradation. These errors arise from a significant downward shift in the threshold voltage distribution of the flash cells. The two read-retry modes successfully adjust the read reference voltages to accommodate this shifted distribution, thereby reducing the post-baking RBER to a level within the error correction capability of the ECC algorithm, even at high P/E cycle counts. For example, while the RBER for Chip B at 1000 P/E cycles (1.5×10^{-2}) is significantly over the ECC error correction capability after the chip is baked, Modes A and B can reduce the RBER by 88.6% (i.e., to 1.7×10^{-3}) and 94.6% (i.e., to 8.2×10^{-4}), respectively.

The read-retry mechanism significantly reduces the number of uncorrectable data chunks after the chip undergoes the baking process. Table 3.3 illustrates the number of uncorrectable data chunks for data with a retention age of four weeks, comparing the default read operation with the available read-retry modes. Two key observations can be made from the table. First, at low P/E cycle counts (e.g., 1000 P/E cycles), Mode B can completely eliminate the uncorrectable data chunks introduced by baking. Second, at higher P/E cycle counts, while the read-retry modes do not fully eliminate the uncorrectable data chunks, they can substantially reduce their number. For instance, at 2500 P/E cycles, the default read operation results in uncorrectable errors in all data chunks, whereas read-retry Mode B reduces the number of uncorrectable data chunks to 49.5% of the total. These results suggest that further improvements in the read-retry mechanism could be achieved by refining the mode to enhance its effectiveness, potentially by adjusting the read voltage more aggressively and improving communication with the flash memory controller.

Table 3.3: Fraction of pages containing uncorrectable 1KB data chunks after Chip B is baked, with and without read-retry.

Read Mode	P/E Cycle Count				
	10	**300**	**1000**	**2500**	**4000**
Default Read	0.0%	83.6%	99.7%	100.0%	100.0%
Read-Retry Mode A	**0.0%**	**0.0%**	12.1%	69.0%	99.1%
Read-Retry Mode B	**0.0%**	**0.0%**	**0.0%**	49.5%	90.6%

Therefore, it is recommended that digital forensic investigators utilize the read-retry mechanisms integrated into NAND flash memory chips. These mechanisms can effectively mitigate the substantial number of uncorrectable errors that arise due to the exposure of chips to extremely high temperatures during chip-off analysis.

3.6 Conclusion

With the increasing popularity of NAND flash memory as a storage medium, digital forensic investigators are often required to perform data recovery from a growing number of NAND flash memory-based devices seized during criminal investigations. This chapter demonstrated that the amount of time that elapses between device seizure and data extraction can increase the error rate of data stored within the device. If the target chip is heavily worn, this error rate may exceed the capacity of the internal error correction mechanisms originally implemented by the NAND flash memory controller, leading to uncorrectable errors. Under normal operating conditions, the flash memory controller continuously monitors the status of the data and proactively re-writes it to new locations before errors become uncorrectable, thereby preventing data loss. However, once the device is removed from normal operation, this corrective process stops, resulting in an increased risk of uncorrectable errors over time.

Therefore, it is critical for digital forensic investigators to perform data extraction from NAND flash memory-based digital devices at the earliest point of time after seizure of the target device. In situations where thermal-based chip-off analysis is required, the high temperature can further increase the error rate by more than two orders of magnitude, despite the use of best practices taken from electronics rework procedures. It is demonstrated that using the read-retry mechanism built into modern NAND flash memory chips, instead of simply using the default read operation when extracting data, provides a promising solution. The read-retry mechanism can significantly mitigate the error rate increase caused by the thermal-based chip removal process.

In conclusion, when handling NAND flash memory for forensic analysis, data needs to be extracted at the earliest possible time, and thermal-based chip-off should be avoided when possible. Otherwise, the read-retry mechanism should be adopted as part of the data recovery procedure in order to maintain the reliability of the extracted data.

4
Data Recovery from Embedded MultiMediaCard

In this chapter, data recovery procedures for embedded MultiMediaCard (eMMC) are explored. eMMC is one of the "managed" flash memory devices widely used in modern digital devices as a storage medium. Consisting of both NAND flash memory and a flash memory controller, eMMC optimizes data input/output between the host device and the non-volatile memory through its standardized protocol. This integration makes eMMC the preferred choice over raw NAND flash memory for mobile device manufacturers. The standardized structure and protocol of eMMC also simplify forensic physical data acquisition compared to handling raw flash memory. However, its secure data purging features, such as Secure Erase and Sanitize, present significant challenges for data recovery. In this chapter, the detailed structures of multiple eMMC devices are investigated, leading to an evaluation of advanced data recovery procedures.

4.1 Embedded MultiMediaCard in Mobile Devices and Its Impact in Forensic Data Extraction

Recovering deleted data for forensic investigations was traditionally performed directly on flash memory, as discussed in Chapter 3. However, as mobile devices have become smaller, memory devices began to be integrated into single components along with their controllers. In modern digital devices, "managed" flash memory is commonly used as the storage device.

These modern digital devices include smartphones, tablets, navigation systems, and various Internet of Things (IoT) devices. Examples of managed memory devices include Solid State Drive (SSD), Universal Flash Storage (UFS), embedded MultiMediaCard (eMMC), Embedded Multi Chip Package (eMCP), and UFS-based Multi Chip Package (uMCP). Among these, eMMCs are currently the most widely used in mobile devices [69, 70, 71].

An eMMC consists of flash memory and a flash controller embedded into a single Integrated Circuit (IC) chip. Its specifications are defined by the Joint Electron Device Engineering Council (JEDEC) [72, 73, 74]. The embedded flash memory stores the data, while its operation is managed by the flash controller. Directly controlling flash memory requires various data optimization processes, including error correction and crafting various vendor-specific commands (see Chapter 3). However, with the integrated flash memory controller, the host system can manage the memory by issuing standardized commands. This allows forensic investigators to acquire the entire image of the target device through the eMMC simply by issuing standardized commands through its interface, resulting in an image that has already been optimized by the flash controller.

While the standardized protocol simplifies physical acquisition compared to handling raw flash memory directly, the structure of the eMMC presents two major challenges for forensic data recovery. First, since the internal flash memory is inaccessible through the eMMC interface, data equivalent to traditional physical data cannot be easily acquired. The data obtained via the eMMC interface is the result of processing by the flash controller, and the acquired image is typically in a format that the host system can directly recognize [75]. As a result, even through chip-off analysis, forensic examiners can only obtain logical data, which includes file system data but excludes unallocated space and low-level raw data. Second, multiple erase commands are implemented in eMMC specifications, such as Secure Erase, which is designed to immediately erase data in the target addresses, along with any copies. Once this command is executed, the erased data becomes completely inaccessible through the eMMC interface, significantly reducing the forensic data recovery rate.

Forensic data recovery typically involves accessing and carving unallocated data on the storage media of the target device [76, 77, 78, 79, 80]. When data is *logically* deleted, it is flagged as deleted by the system, but the actual data may remain on the storage media. Forensic examiners often attempt to recover deleted data by acquiring a physical image of the storage media. To acquire all the raw data on the storage device, forensic examiners can turn to physical data extraction using chip-off analysis. Chip-off analysis against NAND flash memory directly provides the entire dataset, offering a greater chance of recovering

deleted data by examining the whole system data [69]. However, in the case of eMMC, due to the presence of its internal flash memory controller, the data acquired through chip-off is equivalent to logical data, specifically file system data. Therefore, the acquired data differs from the physical data discussed in Section 2.2.1 in Chapter 2.

The goal of this chapter is to access and recover data from the physical memory *inside* the eMMC. Since the internal physical memory inside an eMMC is not accessible through normal means, reverse-engineering the hardware structure of eMMCs is required. While the hardware specification is standardized for managed flash devices, the internal structure varies by model and manufacturer.

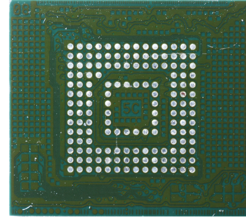
4.2 Embedded MultiMediaCard Structure

An eMMC is typically packaged into a Ball Grid Array (BGA) IC package with dimensions of 12mm×16mm, 12mm×18mm, 14mm×18mm, or 11.5mm×13mm. Figure 4.1a and Figure 4.1b show the top and bottom views of an eMMC in the 11.5mm×13mm package, respectively. Of the 153 ball connectors shown in Figure 4.1b, 30 are assigned to standardized signals, including supply voltage and ground lines (annotated as VDD/VDDF and GND in Figure 4.1c, respectively). The assigned pin names are shown in Figure 4.1c.

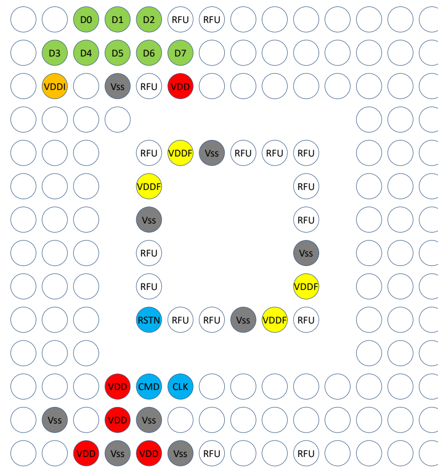
When controlling an eMMC, the host system communicates with the eMMC through signal pins. The key signals required for communication are CMD (Command), CLK (Clock), and DAT[0-7] (Data). The CLK is an input signal from the host system, while CMD and DAT are shared bidirectional signals between the host controller and the eMMC. To control an eMMC, the host system sends a 48-bit command on the CMD line, with each bit of the command sampled at the rising edge of the clock signal on the CLK line. In response to commands from the host system, the eMMC sends a response on the CMD line, which can be either 48 bits or 136 bits in length. When reading and writing data, the data is transferred on the DAT lines after the command is acknowledged by the eMMC. The data bus width is configurable to either 1 bit (using only DAT[0]), 4 bits (DAT[3:0]), or 8 bits (DAT[7:0]). A total of 64 commands are available for use [73]. In addition to the standard commands, manufacturers can implement vendor-specific commands, typically for special operations such as smart report and RAM read/write [81].

An eMMC integrates flash memory and a flash memory controller into a single package, as shown in Figure 4.2. The flash controller manages data read and write operations on the non-volatile flash memory, interpreting commands received via the standardized eMMC interface. Traditionally, flash memory and memory controllers have been packaged as sep-

(a) eMMC Top View



(b) eMMC Bottom View



(c) eMMC standard pinout (top view)

Figure 4.1: Example of an e·MMC in 11.5mm×13mm Package. An e·MMC can be controlled by 10 signals (CLK, CMD, DAT[7:0]). Each signal is assigned to one of the 153 ball connector pads that are visible in 4.1b. The detailed pin assignment is shown in 4.1c

arate IC chips, requiring consumer device vendors to include both components in their product designs. Additionally, flash memory control protocols varied between vendors and products. The integration of two components simplifies device design by combining storage and control functions into a single IC, reducing the need for separate components and varied protocols. The following subsections delve into the detailed structure and functionality of the flash memory and its controller within an eMMC.

4.2.1 Flash Memory Controller

As discussed above, the flash memory controller in an eMMC is responsible for managing communication between the host device and the flash memory. It optimizes this interaction and handles read/write operations to the internal flash memory. In addition to

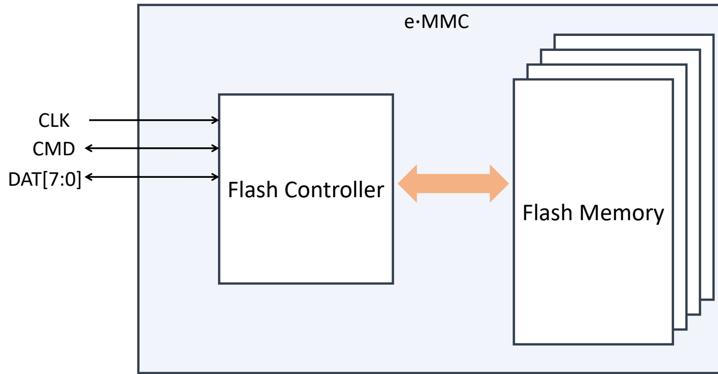


Figure 4.2: The illustrated structure of an eMMC. eMMC interface signal pins are connected to the flash controller, which is responsible for controlling flash memory. Note that flash memory may consist of multiple memory dies.

functioning as a Flash Translation Layer (FTL) [82], the flash memory controller performs several other operations to enhance performance and ensure data integrity. From a digital forensic standpoint, the following operations performed by the flash memory controller are particularly significant, as they alter the original data before storing it in flash memory:

- *Error Correction:* As discussed in Chapter 3, raw NAND flash memory data is prone to bit errors. To address this, flash memory controllers incorporate error correction capabilities, as specified by the flash memory vendors. These error correction schemes and their parameters vary by vendor and model. When storing data, the NAND flash memory controller computes and stores Error Correction Code (ECC) alongside the actual data to detect and correct errors.

- *Data Randomization:* Data is stored in flash memory cells as electrical charge, arranged in pages and blocks (Section 3.2.1 in Chapter 3). Low entropy in data can create imbalanced charge states between neighboring cells, increasing susceptibility to bit errors. To mitigate this cell-to-cell interference, the flash memory controller randomizes the data before storage [35]. This process avoids storing identical data consecutively across flash memory addresses. The controller typically achieves data randomization by computing the XOR value between the data and pseudo-random data, often generated using a Linear Feedback Shift Register (LFSR). The LFSR's structure and initial value vary by model.

- *Wear Leveling:* The bit error rate in flash memory increases with the number of program/erase cycles (Section 3.3.1 in Chapter 3). To prolong the memory's lifespan, the flash memory controller performs wear leveling by normalizing the write/erase counts across physical memory addresses. It records logical addresses along with the data to help translate physical addresses into logical image addresses.

The original data from the host system undergoes these operations before being stored in flash memory. Consequently, recovering the original data for digital forensic purposes requires processing the extracted data. Typical recovery procedures include: correcting bit errors with ECC, de-randomizing data using the LFSR, and sorting data using logical block numbers. Identifying the necessary parameters for these operations often involves mathematical analysis, as reported by Zandwijk [37].

4.2.2 Flash Memory

Flash memory stores the actual data after it has been processed by the controller, as described in the previous section. Flash memory can typically be controlled by the signals outlined in Table 4.1 [83].

Table 4.1: NAND flash memory basic signals

Pin Name*	Input/Output	Description
WP_n	Input	Write Protect
ALE	Input	Address Latch Enable
CE_n	Input	Chip Enable
WE	Input	Write Enable
RY/BY_n	Output	Ready/Busy
RE_n	Input	Read Enable
CLE	Input	Command Latch Enable
I/O[7:0]	Input/Output	Data Input/Output

* "_n" after the signal name means that the signal is active low. Therefore enabled when the signal is pulled down to ground.

Multiple CE_n and/or RY/BY_n signals may exist in one eMMC. When multiple flash memory dies* are present, the CE_n signal is used to select the target die, while other signals are typically shared among all dies.

Flash memory operations are performed at two levels of granularity:

- *Page Operations:* Data is read from or written to flash memory in units called pages. A page is a group of memory cells connected in serial, and its size varies by model, typically ranging from 4K to 16K bytes in modern flash memory.

*A die is the individual physical piece of microfabricated semiconductor material.

- *Block Operations:* Data erasure occurs at a larger unit called a block, which consists of multiple pages. The number of pages per block can vary, typically including 128, 256, or 512 pages, depending on the flash memory structure.

Due to the difference in granularity between page-level writes and block-level erases, erasing data in flash memory is not immediate. When the host system issues a data erase command, the data in the targeted block is not erased right away. This is because a block that contains erased data may also hold valid data in other pages. To erase data at the flash memory level, the flash memory controller first relocates any valid data from the target block to another block, then erases the entire block. As a result, erased data may still remain in the flash memory until the block is fully erased.

4.2.3 Erasing Data on Embedded MultiMediaCard

Several commands in the eMMC standards perform data erasure, allowing the host system to delete data directly from the target eMMC [73, 72]. These operations typically involve a sequence of commands: defining the start and end addresses with CMD35 (ERASE_GROUP_START) and CMD36 (ERASE_GROUP_END), followed by executing the erase operation with CMD38 (ERASE). The specifics of each operation are determined by the argument provided with CMD38:

- *Erase:* This command initiates a standard erase operation. No argument is provided with CMD38, and the controller may delete the data immediately or at a later, more convenient time.
- *Secure Erase:* Similar to the standard erase, but with CMD38 set to argument 0x80000000, the eMMC controller performs a Secure Erase operation immediately, purging both the specified data and any of its copies.
- *Trim:* The Trim command uses CMD38 with argument 0x00000001, overwriting the target data with zeros or ones. The operation can be executed immediately or deferred, but it does not guarantee secure deletion, so it is often used with Sanitize.
- *Secure Trim:* This operation is more complex, requiring two stages. In the first step, CMD35 and CMD36 mark the target addresses, followed by CMD38 with argument 0x80000001, flagging the data for trimming without immediate deletion. In the second step, CMD38 with argument 0x80008000 completes the secure purge, ensuring that all marked data and its copies are erased. Secure Trim is more efficient than Secure Erase for multiple fragmented erase groups [84].

- *Sanitize:* Introduced in eMMC version 4.5, the Sanitize command physically removes data from the unmapped user address space. The operation begins by writing a non-zero value to the "SANITIZE_START" register of the extended CSD (Card Specific Data) using the SWITCH (CMD6) command. Sanitize is recommended in combination with Trim for secure data removal.

According to the JEDEC standard [73], eMMC version 4.51 and higher should utilize Sanitize with Erase or Trim instead of Secure Erase and Secure Trim to ensure complete removal of data from unmapped user address space.

4.3 Extracting Data from Flash Memory in Embedded MultiMediaCard

Since the physical data of an eMMC is stored in its internal flash memory, acquiring this data allows for analysis of remnants after a data purge operation is performed through the eMMC interface. The data recovery procedure typically follows the steps shown in Figure 4.3.

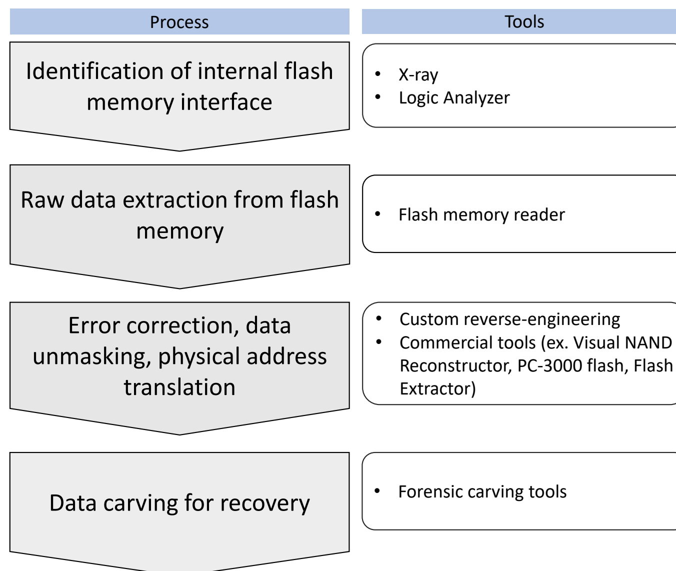


Figure 4.3: Typical workflow of data recovery from inside an eMMC. Required processes are shown here with popularly used tools.

For analyzing an eMMC whose structure is unknown, the procedure typically begins with identifying the internal physical structures, such as physically locating the flash memory and the controller. This step is critical because it allows forensic examiners to under-

stand the layout and connectivity of the components, which is necessary for subsequent data extraction.

After identifying the physical structure, the next step is examining the logical signals to ensure proper connection between the target device and a flash memory reader. This step is crucial because the reader only functions correctly if the correct signals are connected to the appropriate pins; otherwise, the commands issued by the reader cannot be understood by the flash memory controller in the eMMC.

Once the raw data is extracted via the flash memory interface, the forensic examiner must process the physical data to correct bit errors, unmask randomization, and re-order the data to align with the logical addresses. This step poses specific challenges, as each eMMC model has different setups for these operations, requiring detailed research and analysis using reference devices to accurately retrieve the clear data.

4.3.1 Identifying Flash Memory Signal Connections

The first step in accessing the flash memory inside an eMMC is identifying the traces that connect the signal lines of the flash memory dies. During the eMMC packaging process, the flash memory die and the flash controller die are connected to the Printed Circuit Board (PCB) through bonding wires, and then the device is encapsulated as an IC with epoxy molds. An example of cross-sectional view of an eMMC is illustrated in Figure 4.4.

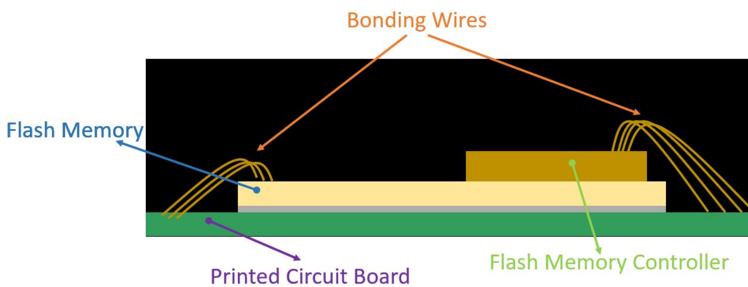


Figure 4.4: Image of the cross-sectional structural view of an eMMC. One or more flash memory dies and the flash memory controller die are connected to the PCB through bonding wires.

The metal bonding wires and the PCB can be visually inspected using X-ray radiography. Figure 4.5 provides an example of an X-ray inspection image of an eMMC, where only dense materials such as metals are visible. In this X-ray image, taken from the top side of the eMMC, the bonding wires appear as thin, straight lines. By tracing the PCB traces to which each bonding wire is connected, an examiner can determine the connectors through which the flash memory is accessed.

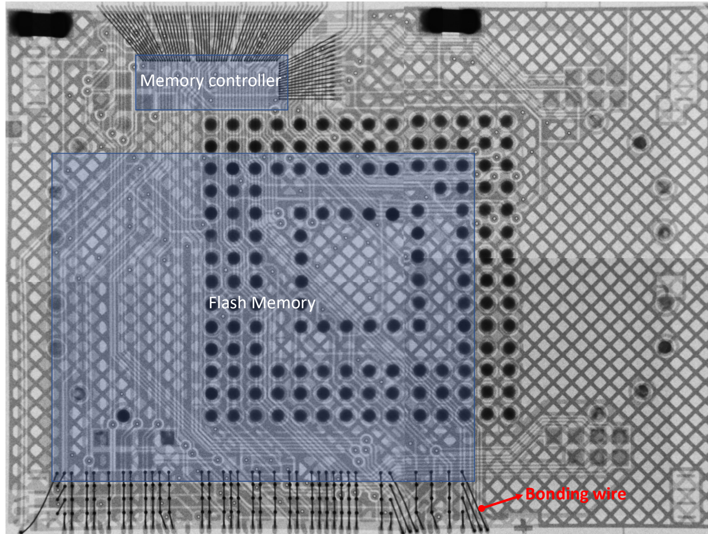


Figure 4.5: X-rayed image of an eMMC. Bonding wires, traces in the PCB, electrical components, and ball connectors are visible. The imaginary locations of the flash controller and flash memory are illustrated.

Caution is required when using X-ray for this purpose. X-ray exposure can potentially damage the flash IC and affect the charge states of the flash memory cells, which might compromise the stored data [85, 86]. Thus, X-ray inspection should be conducted with a reference device for forensic purposes to mitigate potential damage.

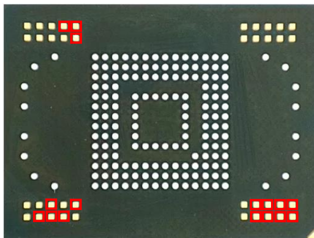
Once the physical paths of the bonding wires have been traced, the signal names of each pin can be determined using a logic analyzer. By monitoring the signal bus between the flash memory controller and the flash memory with a logic analyzer while the target eMMC operates, an examiner can capture and analyze the commands and responses. Comparing these results with known command and response patterns (such as chip ID reads) allows the examiner to map each pin to its corresponding signal, as listed in Table 4.1.

Through these procedures, the connections to internal flash memory in eMMCs from various manufacturers have been identified. Typically, flash memory signal pins are connected to the pads visible on the bottom side of an eMMC package. The locations of these flash signal pins across different eMMC models are detailed below.

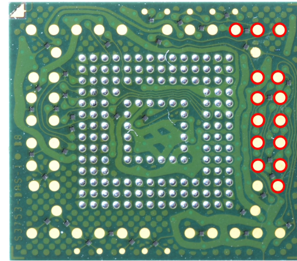
- Samsung
  Figure 4.6a shows the bottom view of Samsung KLMAG2GE4A. Notice that there are multiple access pads at every corner of the package, which are not included in the ball connector pads shown in Figure 4.1c. By tracing the internal connections of the flash memory, it is found that all the flash signal pins are connected to these non-standard access pads. Specifically, the pads indicated in red squares are connected to

the signals listed in Table 4.1.

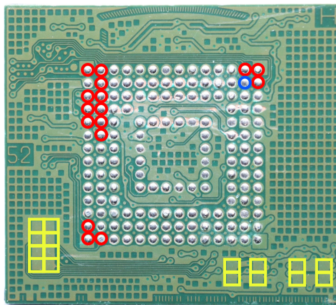
Flash memory in Samsung eMMCs can typically be accessible through these non-standard pads at the corners. The location and number of these non-standard pads vary by product. Therefore, tracing flash signal pins is necessary each time a forensic investigator analyzes a different Samsung eMMC model, along with "bus-sniffing" through a logic analyzer to determine which signal pin connects to each pad.



(a) Samsung KLMAG2GE4A bottom view



(b) Sandisk SDIN7DU2 bottom view



(c) Toshiba THGBMBG7D2KBAIL bottom view

Figure 4.6: eMMC bottom views from multiple manufacturers. Non-standard pins are exposed on Samsung and SanDisk eMMCs.

- SanDisk
  Figure 4.6b shows the bottom view of SanDisk SDIN7DU2. Non-standard access pads surround the standard 153-pin connectors. The flash memory connections can be traced to these pads. It appears that SanDisk eMMC devices typically follow the same flash memory footprint across all models. The access pads indicated with red circles are connected to the internal flash memory.

- Toshiba
  In Toshiba eMMCs, non-standard access pads are typically covered by a coating layer and are only accessible after removing the solder masks. An example pinout is shown in Figure 4.6c, where pins in yellow squares are connected to the internal flash memory. Access to the flash memory can be gained by exposing these pads. Addition-

ally, flash memory in Toshiba eMMCs can also be accessed through the standard 153-pin connector pads. One of the standard connector pads (labeled as N2 in Figure 4.1c and highlighted with a blue circle in Figure 4.6c), functions as a multiplexer input, determining the operational mode of the card. If the N2 pin is connected to the ground (as defined in [74]), the card operates as an eMMC. Conversely, if the N2 pin is pulled high to the input voltage level, the card functions as flash memory. Tracing the internal connections reveals that the pads indicated with red circles in Figure 4.6c are used for accessing flash memory. In Toshiba eMMCs, the data bus lines are shared between the eMMC protocol and the flash protocol.

4.3.2 Flash Memory Data Extraction

Once the locations of all the flash memory signal pins are identified, internal data can be extracted by issuing the appropriate read commands through the identified access pads. Specialized flash memory readers such as the NFI MemoryToolKit [87], Rusolut Visual Nand Reconstructor [88], and Soft-Center Flash Extractor [89] enable examiners to access the target flash memory effectively. These tools facilitate the extraction of raw data across the entire memory region, providing a comprehensive insight into stored data and potential remnants of previously deleted files.

4.4 Data Recovery from Embedded MultiMediaCard

To investigate the data recovery rate from raw flash inside an eMMC, two lab-controlled tests were conducted. The first test involves data recovery from Android smartphones, and the second evaluates the impact of eMMC data purge operations on the internal flash memory. Detailed procedures for each test are discussed in this section.

4.4.1 Deleted Android Data Recovery

Sony Xperia Z2 was selected for data recovery experiments because it stores data in cleartext within the storage media. Six devices, still containing user data, were acquired from the second-hand market. Each device employed a Sandisk eMMC with the model number SDIN7DP2-4G as the main storage medium. The eMMCs were detached from the PCBs of the devices, and their physical images were extracted through the eMMC interface.

Data extraction was performed using the Unix-based `dd` program after connecting the target eMMC to a PC via an SD/MMC card reader. Following the data extraction from

the eMMC interface, the physical data from the internal flash memory was extracted by connecting flash memory access pads of each eMMC to a flash memory reader, as described in Section 4.3.

The acquired flash data underwent error correction and data unscrambling to recover the original plain-text data. To simplify the quantification of the data recovery comparison, only SMS messages were carved out from the acquired data using header information. The number of recovered SMS messages from the eMMC interface and those extracted directly from the raw flash data were then compared.

Table 4.2 presents the total counts of recovered SMS messages from both the eMMC image and the internal flash memory data. The counts of SMS messages recovered from the eMMC represent the maximum number of messages recoverable through traditional physical acquisition performed via the eMMC interface. Comparing these numbers with those recovered directly from the flash memory reveals that some recoverable data still remains in the internal flash memory, not captured through the eMMC interface.

Furthermore, the data recoverable directly from the flash memory includes messages that were either deleted by the user or by the system, as well as those residing in cache data created by the internal flash memory controller [90]. This indicates that more message data is available on the flash memory than what is recoverable through the eMMC, highlighting the limitations of accessing data solely through the host system interface.

Table 4.2: SMS data recovery from eMMC and the internal flash memory

Target	SMS count on eMMC	SMS Count on flash
A	41	116
B	19	47
C	20	105
D	4,723	4,866
E	82	118
F	1,174	1,540

The actual user data recovery rate is influenced by how each application software and the operating system manage deletion operations. Thus, it is impractical to directly extrapolate the user data recovery rate from this experiment alone. Notably, the numbers reported do not account for the overlap between messages recovered from the eMMC and those retrieved from the flash memory, making it unclear how many additional messages are genuinely recoverable.

Nevertheless, the data clearly indicates that exploring the internal flash memory yields

more recoverable data than what is extracted through the eMMC interface alone. For target devices A, B, and C, the amount of user data recovered from the internal memory was more than double that extracted through the eMMC. This result underscores that accessing the internal flash memory of an eMMC significantly enhances the likelihood of recovering deleted data.

4.4.2 Data Recovery after Data Purge Operations

As discussed in Section 4.2.3, multiple data purge operations are defined within eMMC specifications. To examine the impact of these operations on the physical data via the eMMC interface, the same physical eMMC image was duplicated onto multiple eMMCs from various manufacturers, followed by the execution of data purge operations through the eMMC interface. Table 4.3 lists the eMMCs used for these tests. Devices from Samsung, SanDisk, and Toshiba were selected, as these manufacturers' eMMCs are frequently encountered in forensic investigations. Each target eMMC was assigned a unique number, as shown in Table 4.3, and these numbers are used to refer to the devices in the remaining sections of this chapter.

Table 4.3: List of Target eMMCs

Number	Manufacturer	Part Number	Size (GB)	eMMC version
1	Samsung	KLM8G2FEJA-A001	8	4.41
2	Samsung	KLM8G1WEMB-A001	8	4.5
3	Toshiba	THGBM5G7B2JBA1M	16	4.5
4	SanDisk	SDIN7DU2-16G	16	4.41
5	SanDisk	SDIN8DE4-16G	16	4.5

A 4GB Android image data was used as the original dataset, extracted from one of the phones used in the experiment described in Section 4.4.1. This image includes 27 partitions, with the `userdata` partition being the largest, occupying 2.1GB. After confirming the successful data transfer to the eMMC, data purge commands were issued against the `userdata` partition, followed by the extraction of raw flash memory data. *Secure Erase*, *Secure Trim*, and *Sanitize* combined with *Trim* operations were performed to evaluate the impact of each data purge method.

For each test, the raw data extracted through the flash memory interface was compared with the original `userdata` partition stored in the eMMC. This comparison was conducted sector by sector to accurately assess the potential for data recovery. The number of sectors in which data remained on the raw flash memory was counted. The physical sec-

tor address of the `userdata` partition in the eMMC ranges from 0x31E000 to 0x747FDE, consisting of 4,366,302 512-byte sectors. Out of these, 920,717 sectors do not contain data, with the entire block being either all logical zeros or ones. To accurately count the number of recoverable data-containing sectors remaining after the data purge operation, the analysis focused on the 3,445,585 sectors that potentially contained recoverable data.

A custom eMMC programmer was developed using a Raspberry Pi [91], utilizing its General Purpose Input Output (GPIO) pins to control all commands issued to the target eMMCs directly. The programmer's initial task is to initialize the eMMC, setting it up for communication. It then changes the eMMC's operational status to a mode where data transfer is possible. At this stage, the required data purge commands were executed. Using this dedicated programmer guarantees that no unintended operations, such as write commands, would be issued, thus preserving the integrity of the experiment. Additionally, this development ensured that no operating system could interfere with the control of the target chip, allowing for precise command execution.

Before each new experiment, the entire eMMC was overwritten with random data using the command `dd if=/dev/random of=/dev/mmcblk0` to eliminate any influence from residual data. After this, the initial data image was restored on the target eMMC using the `dd` program, followed by the execution of the necessary data purge commands. The Rusolut Visual NAND Reconstructor was employed as the flash memory reader throughout the experiment to acquire raw flash memory data. Figure 4.7 shows the connection setup between one of the target eMMCs and the flash memory reader.

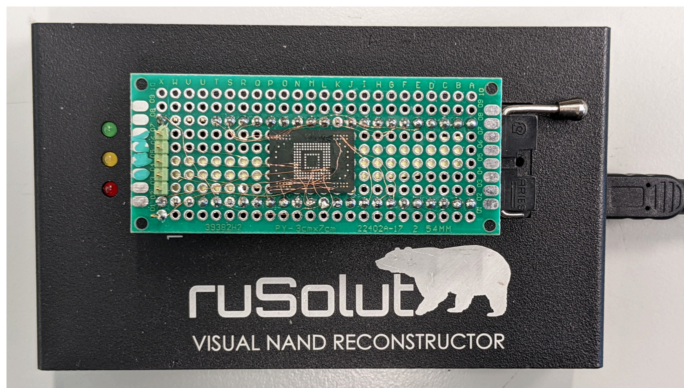


Figure 4.7: Connection between the flash memory interface of a target eMMC and the flash memory reader. The flash memory reader sends the read command to the whole data area of the target flash memory.

Examination of the data through the eMMC interface revealed that the entire `userdata` partition had been overwritten with logical zeros following the erasure operations. Conse-

quently, no user data could be recovered from the data acquired through the eMMC interface after any of these purge operations were applied to the target partition. This implies that if any of these purge operations had been performed on an Android device as described in the previous section, the count of SMS messages recoverable from the eMMC, as listed in Table 4.2, would be zero.

Data recovery rates from the internal flash memory, following each cycle of purge and extraction, are presented in Table 4.4. This table lists the number of sectors recoverable through the flash memory interface. The recovery rate is calculated against the total number of original data-containing sectors, amounting to 3,445,585 sectors.

Table 4.4: Recoverable Sectors after Erase Operations

	Number of Recoverable Sectors after Different Erase Operations (Rate(%))		
Target IC	Secure Erase	Secure Trim	Sanitize
1	935 (<0.001 %)	926 (<0.001 %)	NA (operation not supported)
2	3,425,560 (99.41 %)	3,425,032 (99.40 %)	3,424,652 (99.42 %)
3	3,444,362 (99.96 %)	3,444,223 (99.96 %)	3,444,424 (99.97 %)
4	126,033 (3.66 %)	125,966 (3.66 %)	NA (operation not supported)
5	13,376 (0.0039 %)	11,376 (0.0033 %)	10,231 (0.0030 %)

Almost all the data subjected to Secure Erase remains accessible on target eMMC 2 and 3, allowing for recovery of the original `userdata` partition data. Despite the differing definitions of Secure Erase, Secure Trim, and Sanitize, the outcomes of these data purge operations were similar. Remarkably, even after the Sanitize command, which is deemed the most secure data purge operation, a significant portion of the data remains intact in the flash memory of target eMMC 2 and 3. Conversely, when Secure Erase is strictly implemented, as observed in target eMMC 1, 4, and 5, the majority of the data is effectively wiped from the internal flash memory. For these devices, the likelihood of recovering deleted data is considerably reduced if the data purge operation is initiated from the host system.

During the data purge operations, the operation time was monitored by observing the "busy" signal returned from the eMMC. Following the Erase command, the eMMC issues a response and pulls the D0 line low while performing the operation. It was noted that target eMMC 1 and 3 exhibited longer erase operation times compared to other devices. However, there appears to be no direct correlation between operation time and data recovery rate, as substantial data remains recoverable even after prolonged operation times on target eMMC 3.

4.5 Implications and Extensions of Embedded MultiMediaCard Data Recovery in Digital Forensics

The experiments conducted reveal that securely erased data can often still be recovered from the internal flash memory of certain eMMC models. Given the diverse range of eMMC models produced by various manufacturers, extensive testing on a larger scale is essential to develop a comprehensive database of data recovery rates for securely erased eMMCs. Different chip models possess distinct flash controller architectures, which handle internal flash memory data uniquely. Despite this variability, extracting raw data from eMMCs remains a potent approach when a thorough investigation is required. This section will interpret the results within the digital forensic context and explore other environmental factors that could influence data recovery rates.

4.5.1 Forensic Interpretation and Application

Best practices for sanitizing data during an Android factory reset, such as using *ioctl(BLKSECDISCARD)* ([92, 93]), typically involve issuing data purge commands such as Secure Erase (or Secure Trim) or Trim combined with Sanitize, depending on the eMMC version. Although these operations render the erased data inaccessible via the eMMC interface, the results discussed in Section 4.4.2 suggest that a significant portion of the original data often persists in underlying flash memory.

The JEDEC standard allows for “Erase” and “Trim” commands to be executed at the controller’s convenience [73]. Assuming that the flash memory controllers in the target eMMCs defer actual erasure until system downtime, the target eMMCs were kept powered on after data erasure operations, with clock signals active. Despite this, and even after waiting over an hour and performing multiple power cycles, the data recovery rate remained constant.

Additionally, Kim et al. suggests implementing Trim/Discard operations as background tasks to reduce latency [71]. Commands to initiate these background operations were manually sent to the target eMMCs, but the results were unchanged. This indicates that all data erasure operations were completed by the time data recovery was attempted, and no further data deletion occurred later.

While this research confirms that the methodologies discussed are applicable to all devices equipped with an eMMC, the decision to employ these techniques should be carefully considered. The process requires extensive reverse-engineering of both hardware and software, making it both time-consuming and costly. Additionally, details about eMMC

controller operations are proprietary and vary by manufacturer, which adds complexity to the process. Given the common backlog issues in digital forensic labs [94, 95], examiners must judiciously evaluate whether the potential recovery of data justifies the significant resources required.

4.5.2 Data Sanitization Issues

The lack of proper data sanitization in the resale of storage devices is a significant concern. According to a report by Blancco and Ontrack [96], 42 percent of storage devices sold in the second-hand market contain recoverable data. Moreover, Schneider et al. [97] highlights that flash memory chips are sometimes recycled from one device to another and sold as new low-cost devices. Frequently, these repurposed memory devices retain data from their previous applications, posing a risk of exposing sensitive information to unintended recipients.

Memory ICs, sourced from a wide array of devices such as Android devices, smart TVs, car navigation systems, and other IoT devices, are often recycled. After their intended lifecycle, these digital devices are processed as e-waste, their memory ICs are dismantled from circuit boards, and subsequently remounted onto "new" devices.

In the case of an eMMC, the device's hardware usage history can be examined through the "smart report", accessible via vendor-specific commands. This report allows the system host to assess the health status of the target, providing insights into the internal flash memory's condition, including the number of bad blocks and the count of program/erase (P/E) cycles. Figure 4.8 shows a smart report from an eMMC mounted on a new USB thumb drive. The report reveals that the maximum erase count for the flash memory blocks reached 196, with an average erase count of 140 across all blocks, indicating the device had been previously used.

```
---Smart Report---
Error Mode: 0xd2d2d2d2
Super Block Size: 0x00200000
Super Page Size: 0x00004000
Optimal Write Size: 0x00004000
Number of Banks: 0x00000001
Bank0 Initial Bad Block: 0x00000004
Bank0 Runtime Bad Block: 0x00000000
Bank0 Remain Reserved Block: 0x00000038
Max Erase Count: 0x000000c4 (196)
Min Erase Count: 0x00000000 (0)
Avg Erase Count: 0x0000008c (140)
Read Reclaim Count: 0x00000000
Optimal Trim Size: 0x00002000
Max Erase Count (SLC): 0x0000009c (156)
Min Erase Count (SLC): 0x00000000 (0)
Avg Erase Count (SLC): 0x00000069 (105)
Max Erase Count (MLC): 0x000000c4 (196)
Min Erase Count (MLC): 0x00000070 (112)
Avg Erase Count (MLC): 0x0000008d (141)
```

Figure 4.8: Smart report of the eMMC mounted on a USB thumbdrive sold as new

As demonstrated in Section 4.4, hidden data can still be extracted from the flash memory within eMMCs, underscoring that the prevalent practice of memory recycling could lead to significant information leakage. Potential attackers could exploit recycled memory devices to recover sensitive data, highlighting the need for robust data sanitization practices.

4.5.3 Other Factors Effecting Data Recovery Rate

While the experimental data recovery rate from eMMCs is promising for forensic analysis, various environmental factors must be considered when attempting data recovery from devices equipped with eMMCs. The actual operating conditions can significantly influence the recovery rate. Key operational factors include:

- **Available Physical Space in the Flash Memory:** Consider a scenario where the eMMC in a target device is nearly full, limiting space for large new files. To accommodate new data, existing files may need to be deleted. Under such conditions, the flash controller is compelled to erase old data from the memory, potentially reducing the recovery rate of erased data. To test this, all blocks of the target eMMCs were overwritten with logical zeros. Subsequent data extraction revealed that only a fraction of the old data (less than 0.1%) remained on the flash memory after overwriting.
- **Encryption:** When data is encrypted before storage on an eMMC, it remains encrypted in the flash memory. Consequently, even if the original data is reconstructed from remnants in the flash memory, decryption is necessary. This process can be complex unless the original data is recovered error-free and the decryption method is known.

- **Data Fragmentation:** In contrast to the experiments discussed in Section 4.4.2, where a complete image was copied to the eMMC just before data erasure, real-world digital device usage involves continuous data write and erase cycles. Each update to an existing file results in new data being written to a new physical address while leaving the old data at the original location. This process leads to data fragmentation across multiple locations within the flash memory. Although recovering complete files from fragmented data may be challenging, investigators might still retrieve useful information, such as metadata, from these fragments.

4.6 Conclusion

This chapter has demonstrated that data erased from eMMCs can often still be recovered by directly extracting it from internal flash memory. The process of transforming flash memory data into error-free, clear text involves intensive investigations due to the varied internal structures and technologies embedded in eMMCs by different manufacturers and across models.

Despite these complexities, raw flash memory data can be extracted from eMMCs without compromising their physical integrity or functionality. The methodologies described herein are applicable across a broad spectrum of digital devices, not limited to but including factory-reset Android devices. This reveals that data deleted by host systems can persist on internal flash memory of eMMCs in a recoverable state.

Furthermore, these findings highlight the necessity for enhanced data sanitization protocols, particularly in environments where the secure deletion of sensitive data is critical. The persistence of recoverable data on eMMCs despite purported secure erasure methods calls for ongoing improvements in storage technology and data management practices. Future research should continue to explore advanced sanitization techniques in newer memory technologies.

5 Exploiting Replay Protected Memory Block Authentication in Trusted Execution Environment

Embedded MultiMediaCard eMMC provides a protected memory area known as the Replay Protected Memory Block (RPMB), which stores important data in an authenticated manner to protect host devices like smartphones from unauthorized access. Modifying RPMB data requires a pre-shared authentication key, preventing unauthorized users from altering the stored data. On the host device's processor, this pre-shared key is generated and utilized exclusively within a Trusted Execution Environment (TEE), restricting attackers from accessing it. This chapter examines the use of the eMMC RPMB in an Android device and demonstrates how RPMB-based security features can be compromised through a combination of software and hardware reverse engineering.

5.1 Replay Protected Memory Block

The Replay Protected Memory Block (RPMB) is a specialized hardware memory partition specified by JEDEC standards in modern storage devices such as embedded MultiMediaCard (eMMC) and Universal Flash Storage (UFS) [73, 98]. It is designed to store data in an *"authenticated and replay protected manner,"* securing the stored information against unauthorized access and modification [98].

Authentication within the RPMB is performed through a shared secret key between the eMMC and the host system. This authentication key is securely programmed into both the host system's processor and the eMMC at the manufacturing stage, prior to the device's shipment from the factory.

Writing data to the RPMB requires authentication via a cryptographic hash function, specifically a keyed Hash Message Authentication Code (HMAC) using SHA256. Figure 5.1 illustrates the fundamental process of accessing the RPMB. The host device issues an RPMB command packaged within a data frame structure. This data frame includes the data intended for writing, the address, and the write counter. Using the pre-shared key, the host device calculates the HMAC over the data frame and sends both the data frame and HMAC to the target eMMC.

Upon receiving the data, the eMMC first checks the write counter, then verifies the HMAC using the pre-shared key. If the calculated HMAC matches the received HMAC, the eMMC writes the data to the RPMB partition and increments the write counter by one. The write counter, a security feature of the eMMC, tracks the total number of successful authenticated write operations to the RPMB. This counter is securely stored within the eMMC in a location inaccessible via the external interface, protecting it from unauthorized manipulation.

The write counter provides protection against replay attacks. If it is not implemented, an attacker can monitor the communication between the host and the eMMC, and then reproduce the same communication at a later time, allowing modification of data stored in the eMMC.

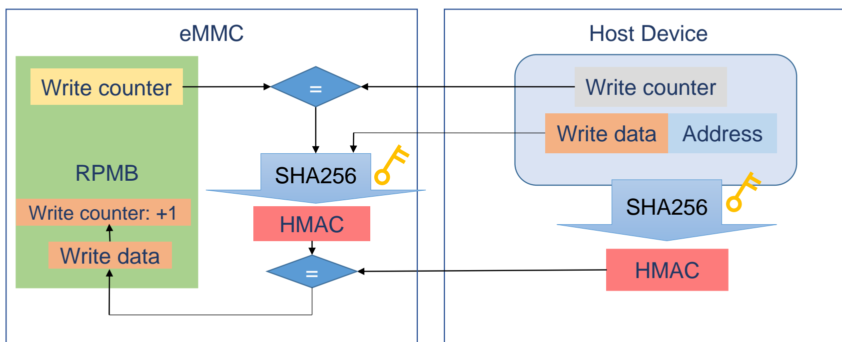


Figure 5.1: RPMB write sequence block diagram.

In modern embedded devices, because RPMB authentication relies solely on the confidentiality of the pre-shared key, a secure component such as a Trusted Execution Environment (TEE) takes ownership of the RPMB [99].

By issuing an RPMB read request command, anyone can read the content of the RPMB data. Therefore, the RPMB is not suitable for storing confidential information. Rather, the RPMB is commonly used to store information that is immutable for a normal user. Example information include version information used in anti-rollback mechanisms [100], cryptographic public keys [101], and the bootloader lock state [102].

Modification of this type of information by an unauthorized attacker can result in an increased attack surface, or in some cases, a fully compromised system. Anti-rollback mechanisms are also often used to prevent software components from being downgraded to a vulnerable state [103, 104].

5.2 Dissecting the Use of Replay Protected Memory Block on an Android Device

5.2.1 Target Device

The Blackphone 2 was selected as the target device to investigate its use of RPMB. The target device was in a wiped state due to multiple incorrect password attempts, and attempts to restore the original pre-wipe filesystem data failed to decrypt the user data successfully.

Testing was conducted on a reference device to validate the occurrence of this issue. The entire filesystem image of this device was acquired before triggering the data wipe and subsequently restored. Although the device booted normally, entering the correct password resulted in a notification stating that while the password was correct, the data could not be decrypted, as depicted in Figure 5.2.

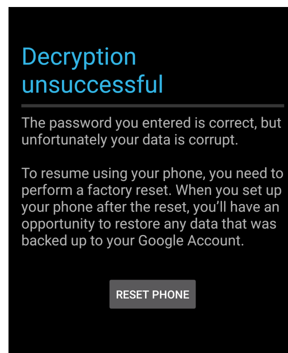


Figure 5.2: The message shown on a test device after triggering data-wipe and restoring the original filesystem image. The devices says the data is corrupt.

Introduced in 2015, the Blackphone 2 is designed with a focus on security and privacy. It features the Qualcomm Snapdragon 615 (MSM8939) System on a Chip (SoC) and runs

on “Silent OS,” a modified version of the Android operating system. The device employs Full Disk Encryption (FDE) to protect user data, relying on both the user’s password and a hardware-bound key to derive the necessary decryption key. The notification displayed on the reference device post-data restoration suggested that the Blackphone 2 employs an anti-rollback counter stored in the RPMB. This counter is a security mechanism designed to prevent the restoration of data backups after a device wipe, effectively blocking efforts to recover sensitive information by potential attackers.

5.2.2 Qualcomm Secure Execution Environment

Qualcomm’s TrustZone technology facilitates the separation of a non-secure operating system (e.g., Android) and a secure operating system, such as Qualcomm Secure Execution Environment (QSEE), on the same device. TrustZone technology is implemented in accordance with the “Advanced Trusted Environment: OMTP TR1” standard [105], which incorporates mitigations against both software and hardware attacks.

The non-secure operating system operates in the Rich Execution Environment (REE), or ‘normal world,’ while the secure operating system runs in the Trusted Execution Environment (TEE), or ‘secure world.’ This separation ensures that certain operations can be securely performed even if an attacker compromises the Android operating system. The secure operating system has full control over the device, whereas the normal operating system can only access the non-secure memory assigned to it. This separation is enforced not only by the Memory Management Unit (MMU) in the application processor but also on the data bus itself by the TrustZone Address Space Controller (TZASC) [106].

Code executed on the application processor (AP) operates at different privilege levels, referred to as Exception Levels (ELs) by ARM. These levels, which dictate the degree of access and control over the processor, are used in both the REE and the TEE as follows:

- EL0: User space
- EL1: Supervisor
- EL2: Hypervisor
- EL3: Secure Monitor

The Secure Monitor facilitates message relay between the REE and the TEE. Only the Linux kernel running at EL1 or the Hypervisor in EL2 are permitted to send messages to QSEE through the Secure Monitor. Consequently, normal applications can only communicate with QSEE through the Linux kernel. Figure 5.3 illustrates the flow of messages

from the Linux kernel in the REE to QSEE in the TEE. On the target device, all critical functionalities related to key derivation and user authentication are implemented within the QSEE kernel itself.

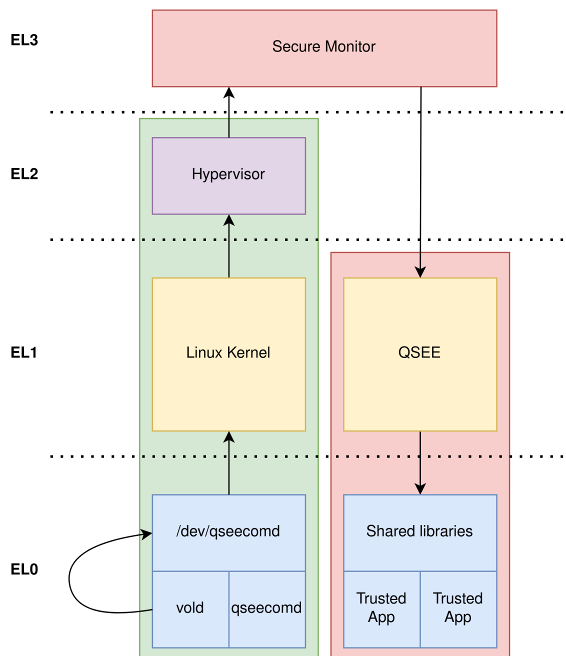


Figure 5.3: A simplified overview of the TZ architecture. Left: REE (Green) Right: TEE (Red)

5

5.2.3 Android Volume Daemon

The Android Volume Daemon (`vold`) is responsible for mounting storage media on Android devices, including the `userdata` partition. The functional process of `vold` was interpreted by analyzing publicly available code repositories that are similar to the operating system of the target device [107]. These insights allowed for a deeper understanding of `vold`'s operations in relation to device encryption and user authentication, which is depicted in Figure 5.4.

Upon user authentication, `vold` initiates communication with the QSEE via the `libQSEEComAPI` library, which provides a secure communication channel through the Linux kernel. This interaction involves sending a request that includes the user password as one of the arguments. The request is processed through a Secure Monitor Call (SMC), managed by the Secure Monitor. Upon receipt, QSEE evaluates the request and returns a response code based on the password attempt status, detailed in Table 5.1.

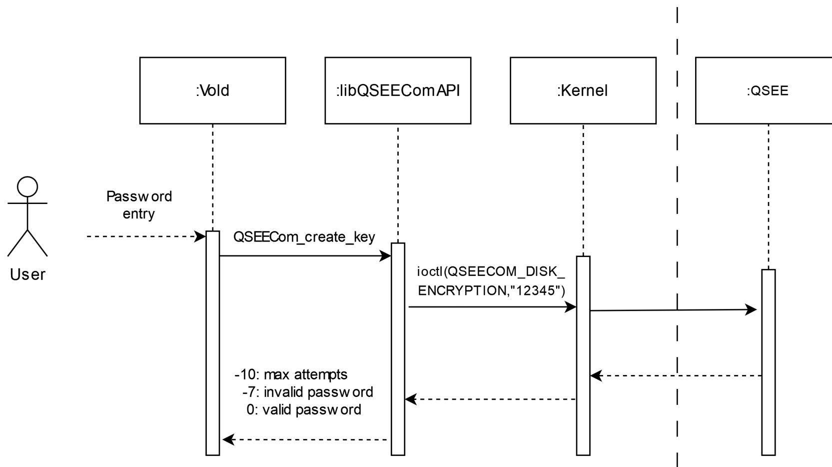


Figure 5.4: A simplified overview of the vold function.

Table 5.1: Response code from QSEE after password attempt

Return value	Decimal	Meaning
0xFFFFFFF6	-10	Max. password attempts reached
0xFFFFFFF9	-7	Invalid password attempt
0	0	Correct password

If the password verification is successful, QSEE configures the decryption key within the crypto engine of the SoC, ensuring that the Application Processor (AP) cannot access the decryption key directly. All interactions concerning the crypto engine are confined to the QSEE environment. The decryption of the `userdata` partition only proceeds after the correct decryption key has been securely established.

5.2.4 RPMB Usage in User Authentication and Key Derivation

Further software reverse engineering identified the use of RPMB on the target device. The QSEE uses a structure known as the *keystore* to securely store sensitive security information, including the FDE key and the count of failed password attempts. Located on a proprietary partition named *SSD* in the Android filesystem, the keystore's data is encrypted with a key derived from a hardware-bound key. If the keystore is not pre-loaded in memory, it is first read from the device's memory storage. QSEE then accesses the RPMB data, using the HMAC included in the RPMB response to validate the stored data. Each entry of the keystore includes an HMAC that ensures the integrity and authenticity of the entry data.

The HMAC is computed over the encrypted data. Authentication fails if the HMAC does not match due to an incorrect anti-rollback counter or a wrong password. When successful, authentication allows the encrypted entry data to be decrypted using a key derived from the hardware-bound key.

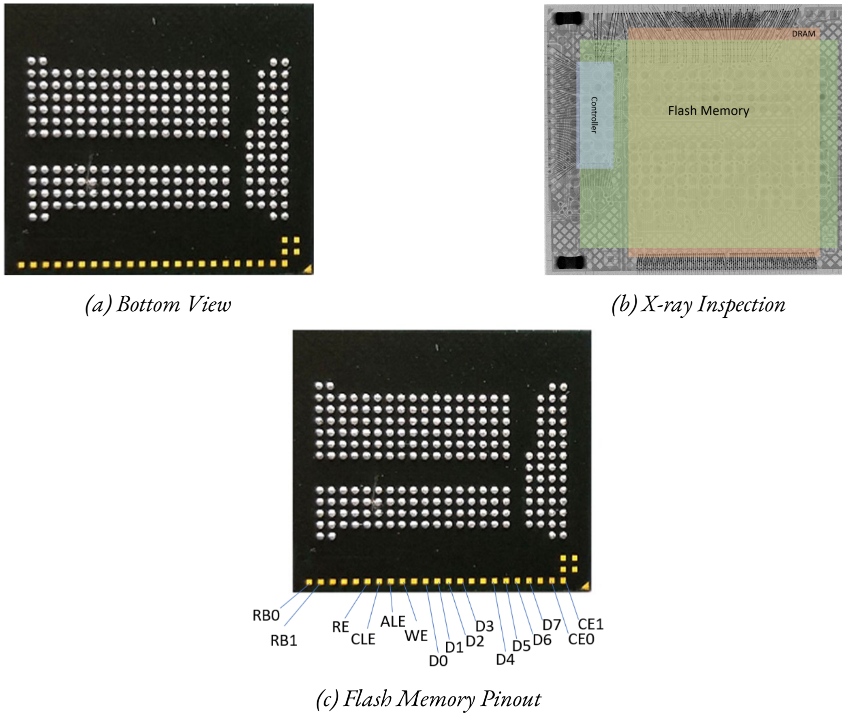
When incorrect passwords are entered more than thirty times, the SoC sends the value 0xFFFFFFF6 (Table 5.1) which triggers QSEE to remove the FDE from the keystore. QSEE then re-encrypts the keystore and writes it back to the flash memory. At the same time, it increments the anti-rollback counter and updates this value in the RPMB. This process ensures that attackers cannot revert the keystore data to an earlier version, as the entries it holds can no longer be authenticated due to the updated counter. Therefore, the decryption mechanism of the target device relies on data stored in the RPMB. Consequently, even if an attacker restores the entire Android OS image to the eMMC, the RPMB remains unchanged, resulting in a mismatch in the key-derivation process and failing the decryption of the user data partition. As noted, the RPMB data cannot be modified without the authentication key, reinforcing its role in securing user data.

5.2.5 Hardware Reverse Engineering for Replay Protected Memory Block Key Extraction

At this point, it becomes evident that manipulating the data stored in the RPMB is necessary to roll back the state of the target device, necessitating hardware-level reverse engineering of the eMMC. The target device utilizes a Hynix eMCP H9TQ26ADFTMCUR for storage, which integrates eMMC and DRAM into a single package. For the remainder of this chapter, the integrated assembly of the flash memory controller and flash memory will be referred to as eMMC, unless otherwise specified.

Following the procedures detailed in Section 4.3 of Chapter 4, the internal structure of the target eMCP was examined using X-ray radiographic inspection. Several eMCPs with the same part number as the target device were acquired to facilitate reverse engineering efforts. Figure 5.5b displays the X-ray image of the target chip, with annotations marking each silicon die's location. Electrical path tracing revealed that the flash memory controller and DRAM are connected to the standard interface, denoted by the silver connector pads in Figure 5.5a. The internal flash memory dies are linked to gold access pads [100]. The pin-out for those access pads were identified as indicated in Figure 5.5c.

To monitor the flash controller of the eMMC behavior during the programming of the RPMB authentication key, the access pads were connected to a logic analyzer, as shown



(a) Bottom View *(b) X-ray Inspection*

(c) Flash Memory Pinout

Figure 5.5: Hynix H9TQ26ADFTMCUR visual observation

in Figure 5.6. This setup captured the communication between the flash memory controller and flash memory during the authentication key programming phase. Using the `mmc-util` program with the `rpmb write-key` option [108], a test key data sequence "12345678901234567890123456789ABC" (in ASCII) was programmed into a reference eMCP chip. The logic analyzer was configured to initiate capture when the Command Latch Enable (CLE) pin of the flash memory was activated.

The captured results, depicted in Figure 5.7a, show multiple communications between the controller and flash memory. Initially, the controller issues several read commands to different addresses in the flash memory, presumably to check the current status of the RPMB. Following this, the controller begins writing values to flash memory. For instance, in the area highlighted in red in Figure 5.7a, the issued command becomes apparent, as detailed in Figure 5.7b. First, with the Command Latch Enable (CLE) pin activated, command 0x80 is issued followed by the address data with the Address Latch Enable (ALE) pin activated. According to the ONFI standard [83], command code 0x80 is used for a page program operation, which in this case is targeted at flash memory page address at 0x0E00000000, as illustrated in Figure 5.7c.

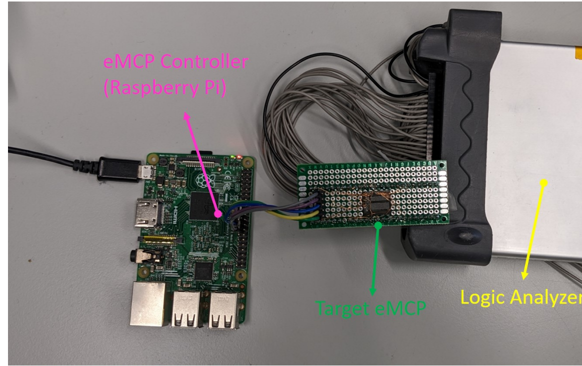
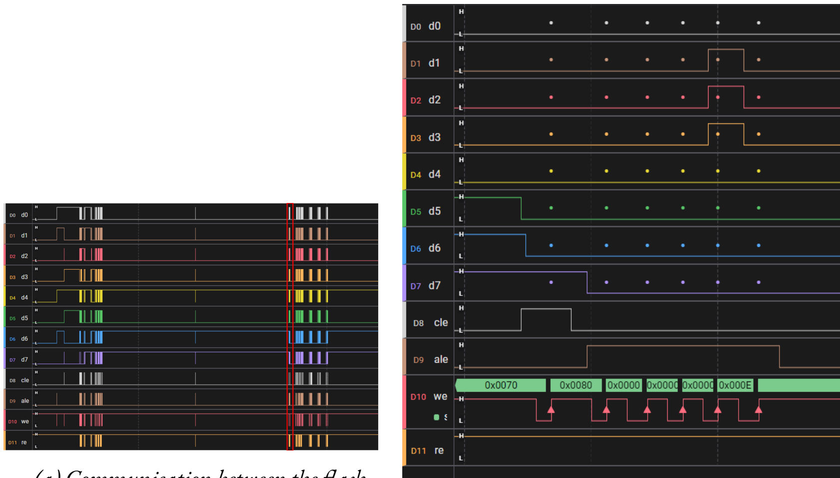
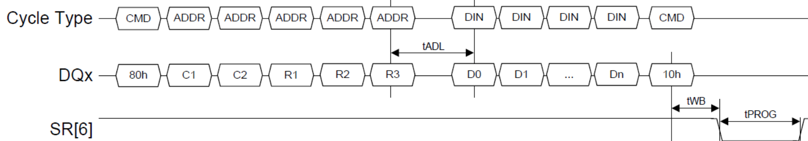


Figure 5.6: Hardware setup to monitor communications between flash memory and the flash memory controller.



(a) Communication between the flash memory controller and flash memory in the eMMC

(b) Zoomed in flash memory command captured during the authentication key programming



(c) Data write command timing defined by ONFI [83]

Figure 5.7: Captured communication between the flash memory controller and flash memory

5.2.6 Physical Memory Dump and Authentication Key Extraction

To ascertain the specific data written to flash memory during the programming of the authentication key, the access pads traced from flash memory (Figure. 5.5c) were connected

to a flash memory reader. The reading operation was performed using the Single Level Cell (SLC) mode facilitated by Rusolut Visual NAND Reconstructor [88]. Due to the reliability concerns associated with MLC flash memory as discussed in Chapter 3, manufacturers nowadays enable SLC mode in MLC flash memory. In SLC mode, each cell stores only one bit of data, enhancing reliability, especially for crucial system data. The operations in SLC mode are executed via proprietary commands, and thus, their implementation varies across different manufacturers.

The raw flash memory data extracted is initially XORed with a scrambling pattern. This pattern was derived from a reference device configured with all plain-text data set to 0x00. While it is possible to reverse-engineer the scrambling pattern [37], for this experiment, identifying a single page of the pattern sufficed to extract the necessary key information.

Using the obtained scrambling pattern, the plain-text data was successfully recovered. Upon analysis of the de-scrambled data, it was discovered that the authentication key data is stored in plain text following the `[PASS]` flag, as depicted in Figure 5.8.

```
0000000250  50 41 53 53 00 80 00 00 00 00 00 00 00 00 00 00  PASS.□..........
0000000260  FF FF 35 00 00 00 00 00 00 00 00 00 00 00 00 00  □□5.............
0000000270  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000280  00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000290  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000002A0  06 04 02 04 01 04 01 07 01 07 06 03 03 03 01 04  ................
00000002B0  00 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ........@.......
00000002C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000002D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000002E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000002F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000300  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000310  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000320  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000330  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 40  ...............@
0000000340  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000350  43 42 41 39 38 37 36 35 34 33 32 31 30 39 38 37  CBA9876543210987
0000000360  36 35 34 33 32 31 30 39 38 37 36 35 34 33 32 31  6543210987654321
```

Figure 5.8: RPMB key stored plain-text in flash memory.

5.3 Restoring Android Device State by Exploiting Replay Protected Memory Block

The reverse engineering efforts demonstrated that the RPMB data is utilized to store the anti-rollback counter, and that the RPMB authentication key can be extracted by accessing flash memory in the eMMC. Upon obtaining the RPMB authentication key, the RPMB content can be modified, effectively compromising the anti-rollback protection.

To validate these findings and apply them in practical scenarios, an experiment was conducted on a Blackphone 2 involving an arbitrary data-wipe and subsequent data restora-

tion. The data wipe was simulated by making automated incorrect password entries until the device initiated a data wipe, during which data modifications were carefully monitored.

5.3.1 Device Setup

Hardware Setup

To facilitate the required eMMC read and write operations, specific hardware modifications were necessary on the target device. Initially, the eMCP chip was detached from the PCB using a heat gun to melt the underlying solder between the eMCP chip and the circuit board of the target device. This step was crucial to make the flash memory access pads of the eMCP accessible.

Subsequently, the eMCP was connected to a Linux-based computer via an eMCP-SD adapter to examine its contents. The output of the `dmesg` command confirmed that the target eMMC was recognized as *mmc0*, containing four physical partitions:

- *mmcblk0*
  The main partition, with a total size of 29GB, partitioned into 32 logical partitions consisting Android filesystem.
- *mmcblk0boot0*
  4KB in size, all bytes were 0x00.
- *mmcblk0boot1*
  Another 4KB partition, also entirely filled with 0x00.
- *mmcblk0rpmb*
  The RPMB partition, sized at 4KB.

Running `mmc-utils` with the `rpmb read-counter` option indicated that the RPMB had been written 155 times. The `dd` program was utilized to perform image acquisition from partitions *mmcblk0*, while `mmc-utils` with the `rpmb read-block` option was executed to access the RPMB contents. These images were preserved as baseline data.

The flash memory access pads of the target chip were then connected to a flash memory reader to extract the RPMB authentication key, located at the same address as in the reference eMCP as detailed in Section 5.2.6.

To monitor data modifications on the RPMB during the data wipe routine without necessitating a chip-off procedure, essential signals controlling the eMMC were extended while the eMCP was remounted onto the PCB. Specifically, the CLK, CMD, and D0 lines of the eMMC were extended from the traces on the PCB using thin wires.

Software Setup

The software running on the target device was also modified in order to automate the data wipe process. Root privileges are required to enable communication with QSEE through the Secure Monitor, as discussed in Section 5.2.2. These privileges are accessible only when unsigned code is executable. A publicly available vulnerability allows for direct modification of the bootloader to change the device's state to unlocked [109], thereby permitting the execution of arbitrary code with EL0 or EL1 privileges.

A modified boot image initiating an Android Debug Bridge (ADB) shell with root privileges was uploaded to the target device using the *fastboot* mode. This configuration granted root access on the device and facilitated communication with QSEE from user space through *ioctl()* calls.

5.3.2 Initiating Data Wipe

By connecting to the device via ADB, multiple password attempts were made using the command `vdc cryptfs checkpw <password>`. After 30 failed attempts, the wipe routine in QSEE was triggered, prompting the device to reboot and begin erasing data.

After the target device completed the wipe routine, the physical image of the eMMC including the RPMB partition was extracted through the use of extended eMMC signal traces. Figure 5.9 shows the difference in data stored in the RPMB partition before and after the data wiping routine. The value stored at offset 0x20C has been incremented by 1 from 0x10 to 0x11 after the device was wiped.

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
000001E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000200   01 00 00 00 01 00 00 00 50 54 42 4C 10 00 00 00  ........PTBL....
00000210   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000220   D3 C2 0D 30 84 1E 5F 55 00 00 00 00 00 00 00 00  ÓÂ.0„._U........
```

(a) RPMB data before user data being wiped.

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
000001E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000200   01 00 00 00 01 00 00 00 50 54 42 4C 11 00 00 00  ........PTBL....
00000210   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000220   D3 C2 0D 30 84 1E 5F 55 00 00 00 00 00 00 00 00  ÓÂ.0„._U........
```

(b) RPMB data after user data being wiped.

Figure 5.9: RPMB data comparison

The preserved baseline data from the *mmcblk0* partition, extracted as outlined in Section 5.3.1, was restored to the eMMC. After booting the device and entering the password,

the message shown in Figure 5.2 appeared (Section 5.2.1), indicating that the data could not be decrypted. This behavior aligns with the reverse engineering findings, confirming that the decryption operation fails due to a mismatch in the RPMB data.

5.3.3 Restoring Data

To recover the original state of the target device, the original data on *mmcblk0* was restored again. Additionally, using the extracted RPMB authentication key, the RPMB partition data was restored to the state acquired in Section 5.3.1. Upon booting and entering the password, the device successfully decrypted the user data, making the original data accessible. Confirming that incorrect RPMB data hindered the restoration of the original target device to its pre-wipe state, the same restoration procedures were applied to the original target device, which remained in a wiped state. Since no original RPMB data was available, restoration involved decrementing the anti-rollback counter at offset 0x20C by one from its current state, aiming to align it with the expected value stored in the encrypted *SSD* partition of *mmcblk0*. It was observed that booting the device after only restoring *mmcblk0* data resulted in the value at offset 0x20C incrementing by one. Due to multiple restoration attempts, this anti-rollback counter turned out to be increased from 0x0A to 0x14, necessitating multiple efforts to incrementally decrease this value with each data restoration attempts. After multiple operations, the state of the original target device was successfully restored to its pre-wipe condition.

Recall from Section 5.1 that the RPMB tracks successful authenticated write operations through the write counter, using it during authentication to ensure data integrity. By rewriting the RPMB partition data with the RPMB authentication key extracted from the eMMC, the write counter value in the target eMMC was also incremented. Despite this, the modifications went undetected by the SoC, allowing the target device to be restored to its pre-wipe state successfully. The device booted without issues, and the user data was successfully decrypted through correct password.

5.4 Attack Mitigation

5.4.1 Safer Storage of the Replay Protected Memory Block Authentication Key

As demonstrated in Section 5.2.6, the RPMB authentication key can be extracted by accessing the internal flash memory of the target eMMC. This key is stored in plain text without any obfuscation or read protection, rendering it accessible to attackers. Although the flash

memory interface is not externally exposed on eMMCs, it remains accessible through chip-off analysis. It was observed that the authentication key of the target device was stored in the same location as in the reference device and that the key data was duplicated at multiple locations. Similar techniques allowed the successful extraction of RPMB authentication keys from eMMCs of other manufacturers used in various smartphones. Each model stored the authentication key at a unique address, however also in plain text without any read protection.

The acquisition of the RPMB authentication key enables attackers to compromise the device through downgrade attacks. If the RPMB is used to prevent software downgrading, an attacker could manipulate the version information stored in the RPMB to a lower value. This change would allow the attacker to downgrade the software to exploit known vulnerabilities, thereby compromising the device. Thus, ensuring hardware-level security of the authentication key is crucial for maintainig the integrity of RPMB in eMMCs.

According to the JEDEC Standard [73], the authentication key should be stored in a "one time programmable" register that cannot be overwritten, erased, or read. However, this is clearly not the case in real-world implementations. There are also commercially available products where the RPMB key can be deleted and the write counter can be cleared. Based on these findings, it can be argued that the hardware-level implementation of the RPMB is not fully compliant with the JEDEC standard, leaving potential vulnerabilities that could be exploited by attackers to compromise the integrity of the device's secure storage.

5.4.2 Use of the RPMB Write Counter

Authentication of a RPMB data write request is performed by computing an HMAC over the message, incorporating the RPMB write counter. The RPMB write counter is a security feature designed to prevent replay attacks. Therefore, even if the RPMB authentication key is compromised, authentication should fail if the write counter value does not match the expected one.

Following modifications to the RPMB partition, as discussed in Section 5.3.3, the RPMB write counter on the target device was incremented. It was anticipated that the QSEE would detect tampering with the RPMB data due to this increment. However, QSEE routinely uses the write counter value supplied by the eMMC, and requests this value anew if an RPMB write operation fails. Consequently, the RPMB write counter does not significantly contribute to the authentication scheme, enabling arbitrary modifications to the

RPMB data once the authentication key is compromised. If the authentication had utilized the write counter, the attack described in previous sections might have failed.

5.4.3 Hardware Authentication

In addition to not utilizing the write counter value for authentication, there appears to be no hardware authentication implemented on the target device. Specifically, even if the target eMMC was swapped with another eMMC of the same model, this modification went undetected. Initially, a cloned eMCP of the target device was created by copying all physical partitions of the eMMC and matching the write counter, since the initial concern was that the write counter needed to match the expected value. This swap also went undetected by the SoC, providing additional avenues for exploitation. If the SoC had been designed to verify the paired eMMC using, for example, the Card Identification (CID) register, and if the write counter had been checked during the authentication process, the attack would not have succeeded.

5.5 Conclusion

This chapter explored the vulnerabilities of the RPMB from both software and hardware perspectives on a specific Android device model. It demonstrated that the integrity of data stored in the RPMB, often perceived as immutable by users, relies heavily on the confidentiality of a pre-shared symmetric key.

This research revealed some security flaws in the current RPMB implementation, both insecure management of the RPMB authentication key on the eMMC and its handling within software running on the target device. These vulnerabilities enable attackers to bypass RPMB authentication mechanisms and manipulate stored data, including disabling anti-rollback protections. Such capabilities expand the potential attack surface, allowing attackers to downgrade device software with the purpose of exploiting known vulnerabilities to compromise the system.

Furthermore, this study highlights the importance of a comprehensive approach in digital forensics, emphasizing the need to integrate both hardware and software reverse engineering to effectively challenge the security features of modern mobile devices. This holistic approach is crucial for developing advanced forensic techniques that can adapt to the increasing complexity of mobile device security.

5

6 Bypassing Replay Protected Memory Block Authentication Through Fault Injection

Chapter 5 examined methodologies for exploiting the use of the Replay Protected Memory Block (RPMB) in embedded MultiMediaCard (eMMC) by directly extracting the authentication key through hardware-level modifications to the target device. This chapter investigates the resilience of the RPMB against fault injection attacks. The objective is to bypass the RPMB authentication process using fault injection techniques, enabling the writing of arbitrary data into the RPMB without knowledge of the authentication key.

6.1 Fault Injection

The term "fault injection" includes various techniques designed to introduce faults or glitches into a device, leading to unintended behaviors. These methods can be applied through both software [110, 111] and hardware [112, 113, 114]. Common hardware techniques include shorting the power supply (known as crowbar glitching or voltage fault injection), applying electromagnetic pulses, illuminating with a laser beam (laser fault injection), and manipulating the clock signal (clock glitching). Fault injection is particularly useful for bypassing security checks in software that does not possess any (known) vulnerabilities. The unintended behaviors, often referred to as fault primitives, can include skipping instructions or corrupting CPU register values, which may be exploited by attackers.

6.2 Experiment Setup

6.2.1 Electromagnetic Fault Injection

The chosen method for the attack scenario in this research was electromagnetic fault injection, as it does not require additional hardware modifications and allows for localized fault injection. Laser fault Injection, which requires thinning the chip package to expose internal transistors, was not considered due to its intrusive nature. Although the clock signal could potentially be susceptible to clock glitching, it was assumed the external clock signal is not directly connected to the internal core processor of an eMMC.

Given that an eMMC device is powered externally, the core voltage of the controller (Vddi) and memory peripherals (Vcc) can be easily manipulated. Thus, applying voltage glitching to these components was also deemed a feasible approach.

6.2.2 Target Selection

The goal of this research is to bypass the cryptographic authentication of the RPMB in an eMMC. Prior to the actual attack, it is crucial to identify the location of the chip where it is most susceptible to electromagnetic fault injection. For this purpose, Samsung eMMCs were selected as target devices since the firmware of their device is accessible. Target devices are detailed in Table 6.1.

Table 6.1: Target chip property details

Target number	1	2	3
Product name	**KLMAG2GE4A**	**KLMBG2JETD**	**KLM8G1WEMB**
Manufacturer ID	0x15	0x15	0x15
eMMC version	4.41	5.1	5.0
Part name	MAG2GA (1.2)	BJTD4R (5.6)	8WMB3R (0.0)
Part no.	0xc23 (Cortex-M3)	0xc27 (Cortex-M7)	0xc23 (Cortex-M3)
Architecture	0x0F (ARMV7-M)	0x0F (ARMV7-M)	0x0F (ARMV7-M)
Variant	0x02	0x01	0x02
Revision	0x00	0x01	0x00
Controller name	VHX0	VCT0	VPX0
MPU enabled	No	No	No
VTOR	0x40000	0x60000000	0x40000

Previous research indicates that Samsung eMMCs typically incorporate a Cortex-M microcontroller, and their firmware can be accessed via vendor-specific commands. In 2018, Avraham demonstrated the ability to read and write to memory regions of an eMMC con-

troller embedded within a mobile device using these commands [115]. This capability was further revealed by the release of proof-of-concept code publicly available on his repository [116].

Having access to the firmware, while not essential for a successful fault injection attack, provides a deeper understanding of the device's operation, potentially enhancing the efficacy of the attack.

6.2.3 Fault Injection Setup

Each target eMMC was mounted on a custom breakout adapter to eliminate the need for reballing and resoldering. The adapter's socket was soldered onto a specially designed Printed Circuit Board (PCB), which was then fixed onto an XYZ table (Genmitsu 3018 PROver V2) using a custom fixture to ensure precise positioning and repeatability. Electromagnetic pulses were generated using a NewAE ChipSHOUTER (CW520) [117], equipped with a one-millimeter clockwise-wound probe. The physical setup is shown in Figure 6.1.

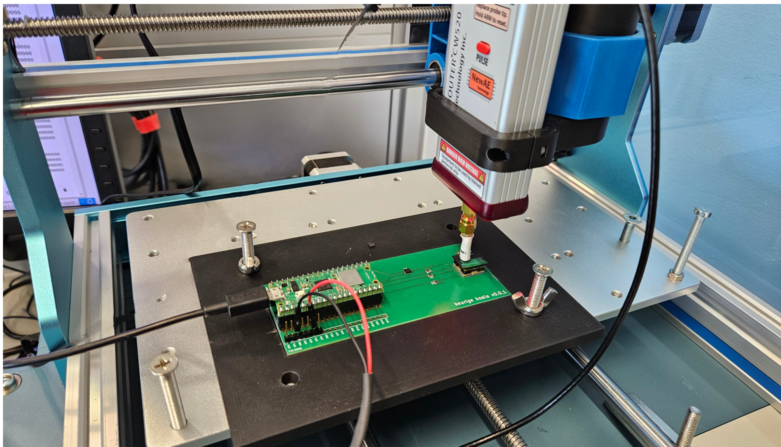


Figure 6.1: Fault injection setup. The target chip is mounted to a custom PCB, using an adapter, while applying an EM pulse using a ChipSHOUTER

The target eMMC was interfaced via programmable I/O (PIO) based state machines running on a Raspberry Pi Pico [118], which also triggered the ChipSHOUTER, initiating pulses at a fixed delay following the eMMC command execution. The whole fault injection process was managed by software running on a Raspberry Pi 4 [91].

A Tektronix DPO7354C oscilloscope was used to monitor the eMMC communication and measure electromagnetic emissions from the internal core processor. These electromagnetic measurements were critical for determining the timing of eMMC operations,

which will be outlined in Section 6.4.1.

While industry level fault injection equipment is available, cost effective off-the-shelf products were used as much as possible for this research.

6.3 Characterizing the Target Against Fault Injection

Electromagnetic fault injection provides the capability for localized attacks. The initial task involved identifying the most vulnerable locations on the target eMMCs. To aid this localization process, a fault observer program was developed and deployed on the target devices, following the procedures outlined in the subsequent sections.

6.3.1 Device Identification

The `mmc-utils` program [108] was extended to include the vendor-specific commands discussed in Section 6.2.2. It was assumed that the internal controller operated based on the ARM architecture. Thus, the extended `mmc-utils` was utilized to read the System Control Block (SCB) at offset 0xE000ED00, and the Memory Protection Unit (MPU) configuration at offset 0xE000ED90. The information acquired from the SCB and MPU is detailed in Table 6.1.

The MPU, which defines the memory access permission, was found to be disabled for all target eMMC units. According to the ARMv7-M Architecture Reference Manual [119], when the MPU is disabled, the default system memory map is utilized. By default, the *Code*, *SRAM*, and *RAM* memory segments are mapped as readable, writable, and executable. The vector table offset register (VTOR) points to the main vector table of the device, with the second entry in this table holding the address of the reset handler, which was used as the entry point for arbitrary code.

6.3.2 Arbitrary Code Execution

The code section also holds the vector table for standard eMMC commands and can therefore be overwritten using vendor-specific commands. For Target 2, this table was located in the RAM segment, but the same principles apply. To gain arbitrary code execution, the payload was first written to an unused memory region, and the entry for CMD8 (SEND_E XT_CSD) in the vector table was updated with the address of the custom routine.

The fault observer implementation was directly written in assembly. Listing 1 shows the decompilation of this code for readability. It consists of a nested `for` loop that increments an unsigned integer value for every iteration. The total number of iterations and the

incremented value are written to the beginning of the extended CSD register, which is currently unused according to the current JEDEC standard [73]. Finally, the original CMD8 routine is executed, returning the contents of the extended CSD register. By checking the stored values, it is possible to determine whether the controller was affected by the EM pulse. This approach worked for all targets.

```
void fault_observer(void) {
  uint32_t total_iterations;
  uint32_t value;
  uint32_t j;
  uint32_t i;
  extcsd *ext_csd;

  ext_csd = PTR_EXT_CSD;
  total_iterations = 0;
  value = 0;
  j = 0;
  do {
    j = j + 1;
    i = 0;
    do {
      value = value + 7;
      i = i + 1;
    } while ((int)i < 62500);
    total_iterations = total_iterations + i;
  } while ((int)j < 4);
  ext_csd->total_iterations = total_iterations;
  ext_csd->value = value;
  (*(code *)CMD8)();
  return;
}
```

Listing 1: Fault observer implementation

6.3.3 Profiling

Electromagnetic pulses were introduced while the fault observer code was running on the target, and the operation was repeated at different positions on the chip to determine which areas were the most affected. Based on the return values of the fault observer, each attempt was categorized as *Normal*, *Crash*, or *Glitch*. If the fault observer returned the expected value, the result was categorized as *Normal*. If the response from the target was all 0x00 or 0xFF, the result was categorized as *Crash*, as the target chip could no longer operate normally without a hard reset. If the result from the fault observer differed from the expected value but the target chip still operated normally, it was categorized as a *Glitch*.

6

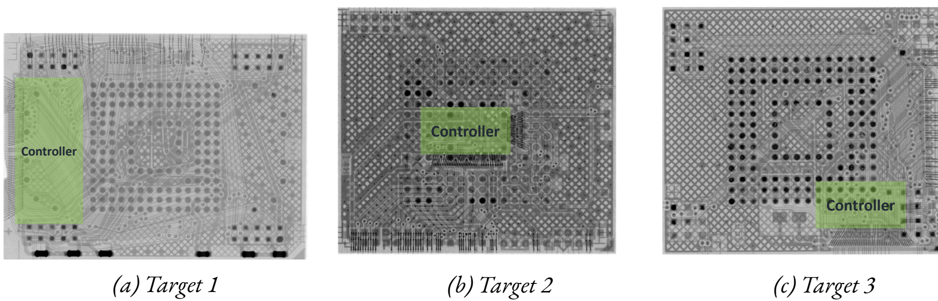
The probe was positioned using a 1mm by 1mm grid overlaid on the target chip. A glitch attempt was performed at each position for 25 iterations. An electromagnetic pulse with a strength of 200V and a duration of 100ns was applied for profiling. A heatmap was created for each categorized result (Figure 6.2).

As shown in Figure 6.2, it is clear that Target 1 is susceptible to the glitching attack at multiple locations. Whereas Target 2 and 3 can be only glitched or crashed if the electromagnetic pulse is sent at the specific location. Nevertheless, through profiling, it is clear that a fault can be injected using electromagnetic pulses, to affect the code running on the eMMC controller for each target. The heatmap was then compared with the internal structure of each target. Using X-ray radiographic inspection, the location of the flash controller in each target chip was identified. Figure 6.3 shows the X-ray image of each target. Based on the bonding wires of the internal controller, the estimated location of the controller is highlighted with a rectangular annotation in the X-ray image.

For Target 1, the location of the "hot spot," where the fault observer can be successfully glitched, matches the location of the controller (Figure 6.2a and 6.3a). Based on this alignment, it is assumed that fault injection is possible if a electromagnetic pulse is sent at the location of the flash memory controller. On the other hand, the hot spot of Target 2 does not match the location of the controller (Figure 6.2b and 6.3b). Rather, this target is more succeptible for *Crashes* and *Glitches* when the electromagnetic pulse is sent directly on top of one of the bonding wires. To the best of available knowledge, this bonding wire is connected to the supply voltage line. It is also worth noting that the controller of Target 2 is located *beneath* the flash memory diesesulting in a greater distance between the electromagnetic probe and the controller compared to Target 1 or Target 3. The fault observation procedure was repeated with higher-voltage EM pulses; however, the results remained unchanged. The hot spot of Target 3 aligns with the location of the controller, though the

(a) *Profiling results: Target 1* (b) *Profiling results: Target 2* (c) *Profiling results: Target 3*

Figure 6.2: Profiling results for each chip using the fault observer. The lighter the color, the more susceptible the chip is for the categorized result at that location.



(a) *Target 1* (b) *Target 2* (c) *Target 3*

Figure 6.3: X-ray inspection of target chips

Table 6.2: Glitching parameters for each target

Target	1	2	3
X position (mm)	154	159.8	159.8
Y position (mm)	-62.5	-58.1	-58.5
Z position (mm)	-25.7	-25.7	-25.7
Pulse Voltage (V)	200	200	200
Pulse Duration (ns)	100	100	100

glitching rate is significantly lower (less than 10 %) compared to Target 1 (approximately 30 %) at the optimal location.

Subsequently, the optimal glitching parameters for each target chip were determined. Different voltages and lengths of the EM pulse were tried at the most susceptible location of each chip (Location x=6, y=4 for Target 1, Location x=10, y=0 for Target 2, and Location x=10, y=1 for Target 3). Pulse voltages and length were selected between 150V and 250V, and between 40ns and 1000ns, respectively. These parameters were randomly chosen while repeating the operation 1500 times. It was observed that Targets 2 and 3 were more prone to crashing when the voltage exceeded 200V. However, the pulse duration did not appear to impact the outcome, as the results were uniformly distributed regardless of pulse length. Consequently, the glitching parameters were selected as shown in Table 6.2. The X, Y, Z position values are based on the setup of the XYZ table.

6.3.4 Firmware Reverse Engineering

To gain a deeper understanding of the RPMB authentication implementation, the firmware of the target eMMC was reverse engineered. All available memory areas, including the boot ROM and main ROM code, were dumped from the chip using vendor-specific commands. The firmware utilizes a vector table located in SRAM, or the RAM segment in the case of Target 2, for all standard eMMC commands. The function responsible for implementing all RPMB functionality was identified and further analyzed to understand how RPMB key authentication could be circumvented using fault injection.

Listing 2 presents the decompiler output for the routine that verifies the HMAC of an RPMB write request.

```
uint32_t rpmb_check_hmac(void *hmac,uint32_t length) {
  uint32_t i = 0;

  if (length + 3 >> 2 != 0) {
    do {
      if (*(int *)((int)hmac + i * 4) != *(int *)(CORRECT_HMAC +
      ↪  i * 4 + 0x60)) {
        return 0;
      }
      i = i + 1;
    } while (i < length + 3 >> 2);
  }
  return 1;
}
```

Listing 2: Routine that checks the RPMB HMAC

The routine checks the HMAC sent in the RPMB write request data frame against the pre-calculated correct HMAC, four bytes at a time. It returns 1 if the HMAC is valid; otherwise, it returns 0. By analyzing the structure of this routine, the following fault injection possibilities were identified:

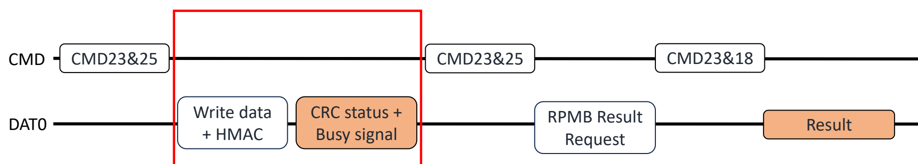
1. If the length argument is set to 0, the check is skipped entirely.
2. If the register `r0` is set to any non-zero value, the ROM assumes the HMAC is valid.
3. If the call to *rpmb_check_hmac* is skipped entirely, the verification will succeed, as `r0` contains a pointer to the provided HMAC (a non-zero value).

6.4 Glitching Replay Protected Memory Block Authentication

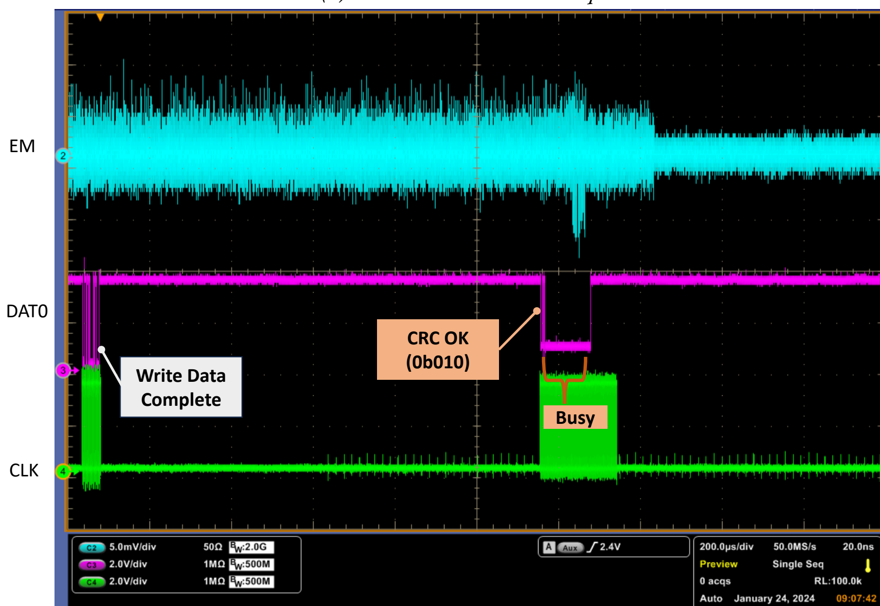
Based on the profiling results of electromagnetic fault injection detailed in Section 6.3.3 and the analysis of the RPMB implementation shown in Section 6.3.4, it was hypothesized that it would be possible to bypass the RPMB HMAC authentication routine using electromagnetic fault injection. Consequently, the setup described in Section 6.2.3 was modified to attack the RPMB authentication.

6.4.1 Identifying the Attack Timing

When writing data to the RPMB, a JEDEC-defined data frame is sent to the target chip. The data frame is 512 bytes long and should include the data to be written, the write counter, address, block count, request message, and an HMAC calculated over this data (see Section 5.1 of Chapter 5). This data frame is sent to the target eMMC following CMD23 (SET_BLOCK_COUNT), which sets the block count and reliable write flag, and CMD25 (WRITE_MULTIPLE_BLOCK), which initiates the block write to the RPMB. The command sequence for the RPMB write request is shown in Figure 6.4a. Commands and data in white boxes are sent from the host to the target, while orange boxes indicate responses from the target. RPMB authentication is likely performed during the time indicated by the red box, making this timing window the optimal target for fault injection.



(a) RPMB write command sequence



(b) Waveforms when data with wrong HMAC were sent to the target. The timing matches the timing window shown in red in Figure 6.4a

Figure 6.4: RPMB write command scheme and captured EM emission

To precisely identify the attack timing window, the electromagnetic radiation emitted by the controller was measured when the RPMB data frame was sent to the target. Figure 6.4b shows the captured waveform. The captured timing aligns with the timing window indicated by the red box in Figure 6.4a. The DAT0 line is shown in magenta, the CLK line in green, and the electromagnetic emission in blue (shown as EM in the figure.) Since significant electromagnetic radiation is observed during and after the data is sent to the target, it is assumed that the controller continues performing internal operations throughout this period. The CRC status of the transmitted data (positive = ”010”) is sent on the DAT0 line from the controller, synchronized with the CLK signal, as defined by the JEDEC Standard. This is followed by the DAT0 line being pulled low, indicating that the target device is busy with internal computations. Even after DAT0 returns to high, the controller appears to remain active, as electromagnetic radiation persists for some time. Since no other commands are issued to the target, it was inferred that HMAC verification is most likely performed during this period, with the observed electromagnetic emissions originating from the controller processing the RPMB data and performing HMAC computation.

6.4.2 Glitching Setup

Prior to the fault injection attack campaign, the RPMB of each target was programmed with an arbitrary key. Then the first block of the RPMB (256 bytes) was programmed with random values. The Raspberry Pi Pico in the setup described in Section 6.2.3 was reprogrammed to communicate with the RPMB on the target eMMC. Then, software running on Raspberry Pi is modified to triger the electromagnetic pulse generation during the timing window identified above. The 200V electromagnetic pulse with a length of 100ns was injected at 10ns granularity during the target timing window. The trigger signal was generated when the last bit of the data packet was sent.

The host system can verify whether the authenticated data write request was successful by reading the result register. This can be done by sending an RPMB data frame with a result register read request, followed by CMD23 and CMD25, and subsequently reading the result by sending CMD23 and CMD18 (READ_MULTIPLE_BLOCK), as shown in Figure 6.4a. Table 6.3 lists the result register values defined by JEDEC [73]. While more values are defined, only the relevant ones for this setup are included in Table 6.3.

The returned register values were used to determine if the fault was successfully injected into the RPMB authentication procedure. If the returned value was 0x02, the target was determined to be responding normally. Since the authenticated data write request was

Table 6.3: Partial list of RPMB operation result register values

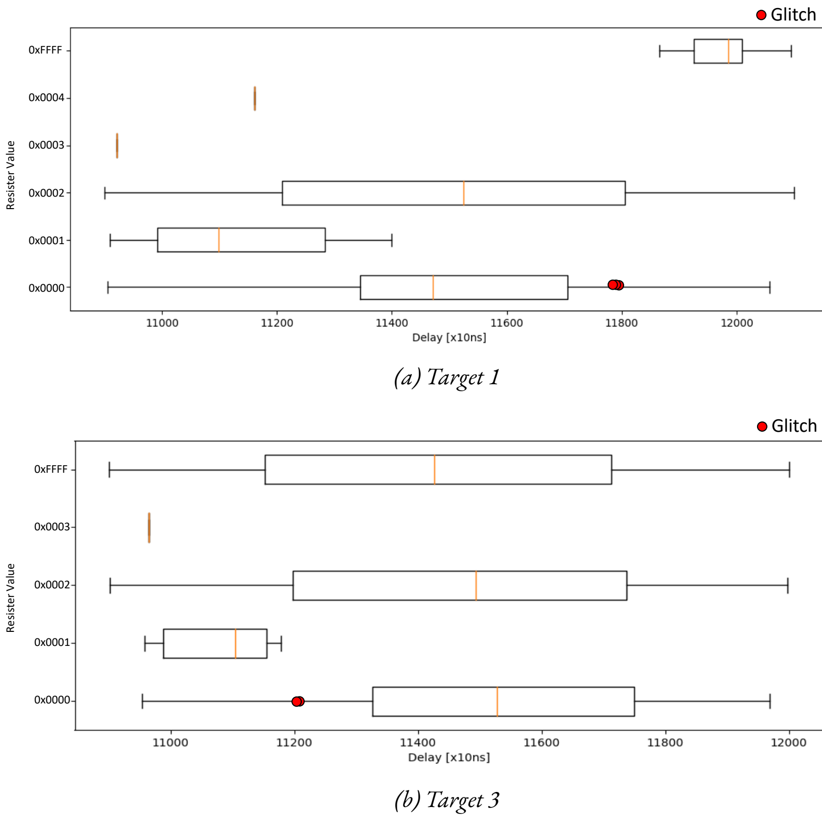
Value	Results
0x0000	Operation OK
0x0001	General failure (Multiple errors have occurred)
0x0002	Authentication failure (HMAC mismatch)
0x0003	Counter failure
0x0004	Address failure

intentionally sent with an incorrect HMAC value, this is the expected result. If the target responded correctly with the register value 0x00, an incremented write counter, and the correct HMAC then the attack has successfully skipped the authentication procedure. If any other value was returned on the result register value, then it was determined that an error occurred during the RPMB authentication procedure. If all of the responses were 0x00 or 0xFF, then the target was presumed to have crashed, since a hard reset is required to bring the chip back to a normal working state.

6.4.3 Results

The glitching campaign targeting RPMB authentication was conducted exclusively on Targets 1 and 3. During the profiling campaign, Target 2 became unresponsive, rendering it unavailable for further analysis. Figure 6.5 shows the returned result register values over time. The x-axis represents the timing of the electromagnetic pulse injection, while the y-axis displays the actual returned value of the result register. The timing is measured as the delay from when the last bit of write data was sent to the target. Responses of 0x0000 and 0xFFFF primarily indicate a target crash, where the entire response data frame was filled with either 0x0000 or 0xFFFF.

On Target 1, the register value 0x0001 (General Failure) was most frequently returned when the electromagnetic pulse was introduced at the early stage of the RPMB operation. Following this, register value 0x0002 (HMAC mismatch) was typically returned if the target did not crash (returned data frame being filled with 0x0000). Register value 0x0003 (Counter Failure) was returned when the electromagnetic pulse was applied at the beginning of the RPMB authentication routine. It is presumed that the write counter check is performed before the HMAC check, and the fault was injected during this routine, resulting in a failure of the write counter check. Similar results were observed on Target 3, with the exception that the crashed state, where the data frame was filled with 0xFFFF, occurred more frequently.



(a) Target 1

(b) Target 3

Figure 6.5: Returned result register value after RPMB authentication glitching

Note that a response 0x0000 is returned in a response data packet if the RPMB authentication was successful, meaning that the target was successfully glitched. Those responses are shown as red dots in Figure 6.5. Based on the results where RPMB authentication was successfully glitched, the critical timing window for fault injection was identified as between 117.72μs and 118.30μs for Target 1, and between 112.30μs and 112.50μs for Target 3. In both cases, these timings occurred near the end of the busy signal indicated on the DAT0 line. Following successful glitching, the write counter value was incremented by one on both targets, indicating that the controller acted as though the correct HMAC had been received and proceeded with writing the data. Post-verification of the RPMB values confirmed that the sent data was successfully stored. Thus, it was concluded that RPMB authentication can be bypassed through electromagnetic fault injection when applied at the correct timing.

6.4.4 Integrity of Non-Volatile Data

While it has been demonstrated that RPMB authentication can be bypassed through electromagnetic fault injection, it is equally important to ensure that the other data stored on the target eMMC remains unaffected. With the critical timing for skipping the RPMB authentication check identified, the experiment was repeated using a new chip. Before the experiment, arbitrary data was written to the main partition of the target eMMC. Prior to initiating the glitching campaign, all physical data was dumped and hashed using SHA-256. The data was imaged using the Unix-based `dd` program.

After completing the non-volatile data extraction, the target chip was mounted on the glitching setup, and the electromagnetic pulse was injected exclusively at the critical timing of the RPMB authentication, as identified in the previous section. For both Target 1 and Target 3, bypassing the RPMB authentication was successful in fewer than 10 attempts (with a success rate of approximately 30% for Target 1 and 10% for Target 3). After successfully bypassing the RPMB authentication, the user data area was extracted again. The SHA-256 hash of the extracted data matched the value computed before the glitching attack. Additionally, the RPMB data was overwritten only at the specified block address, leaving the remaining block data unchanged. Thus, it was concluded that electromagnetic fault injection attacks on RPMB authentication can be performed while preserving the integrity of the stored data, provided that the pulses are applied using predetermined parameters and timing.

However, it must be noted that repeatedly applying electromagnetic pulses to the same device increases the likelihood of data corruption in the user data area of the eMMC. The glitching campaign was repeated 100 additional times using the same setup. As a result, multiple data corruptions were observed in the user data area of both targets, although the RPMB area appeared unaffected. These corruptions were easily reverted by restoring the backup. This observation underscores the importance of creating a data backup beforehand and verifying the integrity of the data after a successful glitching attempt.

6.5 Discussions

6.5.1 Real World Applications

To the best of the author's knowledge, the use of RPMB has seen limited application in smartphones, such as storing an anti-rollback counter (as discussed in Chapter 5). However, this type of storage appears to be more widely adopted in the automotive industry.

For instance, U-Boot, a popular bootloader used in embedded devices, stores anti-rollback counters and the bootloader lock state in the RPMB when Android Verified Boot (AVB) is configured to use OP-TEE [102].

According to NXP, their I.MX family of processors is used by most large car manufacturers [120]. NXP also supports Android Automotive, that uses Trusty as the operating system of choice to run in the Trusted Execution Environment (TEE) [121]. According to the *i.MX Android Security User's Guide* from NXP [122], the RPMB is used to store anti-rollback counters, bootloader lock state, and AVB public key, which is used for verifying integrity of system images. The RPMB key itself is encrypted and decrypted in the TEE by Trusty. The Digi ConnectCore 8X system-on-module (SOM), which is designed around the NXP I.MX 8X processor and uses eMMC storage, also stores the AVB public key in the RPMB [101].

Therefore, it appears that the RPMB is a critical component in the secure boot implementation of a wide range of automotive products. Compromising the RPMB would allow an attacker to rollback to potentially vulnerable software versions, unlock the bootloader, or re-sign system images, ultimately granting the attacker root privileges in the Android operating system.

While an eMMC chip provides tamper-resistant storage through the RPMB, no mitigations against fault injection were observed during the experiments. Additionally, the total cost of recreating the fault injection setup is less than USD $7,000, making electromagnetic fault injection accessible to well-resourced attackers.

6.5.2 Mitigations

As discussed in Section 6.3, publicly known vendor-specific commands were used to achieve code execution on all devices, and run the fault observer routine. This fact already breaks the security of the RPMB, since the firmware can easily be patched through software.

Applying electromagnetic fault injection on an eMMC chip requires physical access to the device. Depending on the threat model being used, this might be a valid concern. One way to increase the difficulty of successful fault injection attacks is to implement mitigations in software as described by van Woudenberg and O'Flynn [123] (e.g. double checking critical data, using non-trivial constants). As mentioned in Section 6.3.4, the implemented HMAC validation routine shown in Listing 2 only fails when returning 0. This requirement is trivial to achieve since the CPU register holding the return value (`r0`) holds a pointer to the HMAC, and thus is non-zero before the function is called. Requiring a

return value with a large Hamming distance (i.e. 0xA5C3B4D2) significantly increases the attack complexity. A large number of bit flips is needed to end up with the correct return value.

Validation of the HMAC is critical; if circumvented, the integrity of the RPMB is fully compromised. Therefore, the HMAC should be checked multiple times. Preferably, a random delay should be added between these checks, requiring the attacker to insert multiple glitches with a non-constant delay between them.

6.6 Conclusion

This chapter demonstrated that it is possible to circumvent the RPMB authentication scheme and write arbitrary data by applying electromagnetic fault injection on eMMC chips. The RPMB functionality in non-volatile storage devices is often relied upon to store critical data that must remain immutable to the user. However, the ability to bypass RPMB authentication and manipulate its data—without knowledge of the authentication key and with minimal hardware modification—presents a valuable opportunity for digital forensic investigators to compromise the target device. This approach could allow them to manipulate data critical to the device's security, such as anti-rollback protection, bootloader lock state, and signature verification.

The implications of this work suggest that while RPMB is widely regarded as a secure element in embedded devices, it is vulnerable to fault injection attacks. This vulnerability provides forensic investigators with a possible entry point for bypassing protections that would otherwise prevent access to critical system areas. By exploiting this flaw, investigators could unlock bootloaders or bypass secure boot processes, enabling them to run arbitrary code on compromised devices.

Future work could explore applying this method to consumer devices and demonstrate a complete attack scenario where the security mechanisms relying on RPMB are fully compromised. Such an attack could allow forensic investigators or attackers to unlock bootloaders and take control of the device, further highlighting the need for stronger fault injection countermeasures to protect the integrity of stored data.

7 Conclusion

This thesis has explored the security features of flash memory-based storage devices from both hardware and software perspectives, with a particular focus on their implementation in modern mobile devices. As discussed in Chapter 2, the advanced security layers in contemporary smartphones mean that traditional data extraction methods, such as chip-off techniques, often fail to yield meaningful results. With the semiconductor industry's aggressive technological advancements, effective memory-based data extraction now requires an in-depth understanding of memory technologies' hardware architecture and their practical applications in mobile devices.

7.1 Addressing the Research Questions

This research addressed the primary research question, **"What kind of security features in flash memory can be exploited to perform effective data extraction from modern mobile devices?"** with the following insights:

- **Current Challenges in Forensic Data Extraction from Mobile Devices:**
  As highlighted throughout this thesis, particularly in Chapter 2, the default encryption of data on mobile devices means that merely extracting physical data from flash memory often does not provide accessible information. Multiple layers of security authentication protect the necessary decryption keys required to decrypt the data stored in flash memory. Consequently, forensic investigators must undertake extensive reverse-engineering to find possible vulnerabilities that allow arbitrary code

execution. This reverse engineering often requires a dual approach—both hardware and software—tailored to each model and software version.

- **Data Storage and Extraction Methods in Flash Memory:**
  Chapter 3 demonstrated that data stored in NAND flash memory is inherently unstable due to the technology's nature, causing bit errors. Additionally, traditional data extraction method, where hot-air is applied to the flash memory chip, can cause critical damage to the stored data. However, vendor-specific commands, like the read-retry command, can mitigate these errors and enhance data reliability. For eMMC, raw flash memory data extraction necessitates hardware reverse-engineering to identify non-standard access pads. Accessing the flash memory allows recovery of deleted data, as detailed in Chapter 4.

- **Security Features in Flash Memory used in Modern Mobile Devices:**
  Chapters 4 and 5 illustrated the advancement of flash memory technology, where flash memory is to be integrated into single components along with their controllers. The eMMC technology, which is most popularly used in modern mobile devices, was the focus of this thesis. One of its security features, the RPMB block, is commonly used in modern storage devices to secure data against unauthorized manipulation. Without proper authentication, data within this block remains tamper-resistant. The RPMB often stores anti-rollback values or hashes necessary for data decryption. Additionally, eMMCs implement secure features such as "Secure Erase" or "Sanitize" to securely delete stored data. Those features are commonly used in modern mobile devices.

- **Exploiting the Flash Memory Security Features:**
  Chapter 4, 5, and 6 demonstrated that above mentioned security features can be exploited to aid effective mobile forensics. By investigating inside the modern memory devices both through software and hardware, those features can be compromised. First, security erase features of eMMCs can be exploited by accessing the internal flash memory on some models, leaving more room for forensic investigators to recover deleted data. Additionally, the implementation of RPMB on modern mobile devices can be exploited through structural reverse-engineering or fault injection techniques, giving vulnerabilities to be exploited. These exploits require extensive reverse engineering and sophisticated techniques to execute successfully.

This thesis also highlighted the fact that both hardware and software-based approaches are critical for performing effective forensic data recovery from modern mobile devices. Keeping up with the current security implementations on these devices requires a dynamic and adaptable forensic methodology. This ongoing challenge underscores the importance of continuous research and development in the field of mobile forensics.

7.2 Future Research

This research focused on NAND flash memory and eMMCs. However, emerging technologies such as UFS and NVMe are becoming the preferred storage solutions for the current generation of smartphones, making them an interesting subject for future research. These devices also incorporate RPMB implementations. Compromising the integrity of RPMB data could allow attackers to further modify the mobile devices, enabling them to take control of the target system. Expanding non-invasive attacks such as fault injection to consumer devices represents a promising avenue for further exploration.

Bibliography

[1] K. Barmpatsalou, D. Damopoulos, G. Kambourakis, and V. Katos, "A Critical Review of 7 Years of Mobile Device Forensics," *Digital Investigation*, vol. 10, no. 4, pp. 323 – 349, 2013.

[2] A. Al-Dhaqm, S. Razak, R. A. Ikuesan, and V. R. Kebande, "A review of mobile forensic investigation process models," *IEEE access*, pp. 1–1, 2020.

[3] P. Reedy, "Interpol Review of Digital Evidence 2016 - 2019," *Forensic Science International: Synergy*, 2020.

[4] M. Chernyshev, S. Zeadally, Z. Baig, and A. Woodward, "Mobile Forensics: Advances, Challenges, and Research Opportunities," *IEEE Security and Privacy*, vol. 15, no. 6, pp. 42–51, 2017.

[5] E. Casey, G. Fellows, M. Geiger, and G. Stellatos, "The Growing Impact of Full Disk Encryption on Digital Forensics," *Digital Investigation*, vol. 8, no. 2, pp. 129 – 134, 2011.

[6] C. Hargreaves and H. Chivers, "Recovery of Encryption Keys from Memory Using a Linear Scan," in *2008 Third International Conference on Availability, Reliability and Security*, 2008, pp. 1369–1376.

[7] J. D. Kornblum, "Implementing BitLocker Drive Encryption for Forensic Analysis," *Digital Investigation*, vol. 5, no. 3–4, pp. 75–84, 2009.

[8] T. Groß, M. Busch, and T. Müller, "One Key to Rule Them all: Recovering the Master Key from RAM to Break Android's File-Based Encryption," *Forensic Science International: Digital Investigation*, vol. 36, p. 301113, 2021.

[9] R. Ayers, S. Brothers, and W. Jansen, "Guidelines on Mobile Device Forensics," National Institute of Standards and Technology, 2014.

[10] C. da Silveira, R. de Sousa, O. Albuquerque, G. Amvame Nze, G. de Oliveira , A. Orozco, L. García Villalba, "Methodology for Forensics Data Reconstruction on Mobile Devices with Android Operating System Applying In-System Programming and Combination Firmware," *Applied Sciences*, vol. 10, p. 4231, 06 2020.

[11] S. Willassen, "Forensic Analysis of Mobile Phone Internal Memory," in *Advances in Digital Forensics*, 2005, pp. 191–204.

[12] M. Breeuwsma *et al.*, “Forensics Data Recovery from Flash Memory,” *Small Scale Digital Device Forensics Journal*, 2007.

[13] F. Courbon, S. Skorobogatov, and C. Woods, “Reverse Engineering Flash EEPROM Memories Using Scanning Electron Microscopy,” in *Smart Card Research and Advanced Applications*, 2017, pp. 57–72.

[14] R. Loftus and M. Baumann, “Android 7 File Based Encryption and the Attacks Against It,” 2017. [Online]. Available: http://delaat.net/rp/2016-2017/p45/report.pdf

[15] P. Teufl, T. Zefferer, and C. Stromberger, “Mobile Device Encryption Systems,” in *IFIP international information security conference*, 2013, pp. 203–216.

[16] Statista, “Android Operating System Share Worldwide by OS Version from 2013 to 2020,” 2020. [Online]. Available: https://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/

[17] T. Vidas, C. Zhang, and N. Christin, “Toward a General Collection Methodology for Android Devices,” *Digital Investigation*, vol. 8, pp. S14 – S24, 2011.

[18] K. Schot and A. Fukami, “In-Situ Global Ultra Thinning of Live Chip Backside for Digital Forensic and Failure Analysis,” in *ISTFA 2023: Conference Proceedings from the 49th International Symposium for Testing and Failure Analysis*, Nov. 2023, pp. 205–208.

[19] A. Vasselle, H. Thiebeauld, Q. Maouhoub, A. Morisset, and S. Ermeneux, “Laser-Induced Fault Injection on Smartphone Bypassing the Secure Boot-Extended Version,” *IEEE Transactions on Computers*, vol. 69, no. 10, pp. 1449–1459, 2020.

[20] R. Hay, “fastboot oem vuln: Android Bootloader Vulnerabilities in Vendor Customizations,” in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Aug. 2017.

[21] V. Katalov, “iOS Device Acquisition with checkra1n Jailbreak,” Nov. 2019. [Online]. Available: https://blog.elcomsoft.com/2019/11/ios-device-acquisition-with-checkra1n-jailbreak/

[22] G. Alendal, G. O. Dyrkolbotn, and S. Axelsson, “Forensics Acquisition — Analysis and Circumvention of Samsung Secure Boot Enforced Common Criteria Mode,” *Digital Investigation*, vol. 24, pp. S60–S67, Mar. 2018.

[23] E. Brier, C. Clavier, and F. Olivier, “Correlation Power Analysis with a Leakage Model,” in *Cryptographic Hardware and Embedded Systems - CHES 2004*, Aug. 2004, pp. 16–29.

[24] A. Vasselle, P. Maurine, and M. Cozzi, "Breaking Mobile Firmware Encryption through Near-Field Side-Channel Analysis," in *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop*, 2019, p. 23–32.

[25] O. Lisovets, D. Knichel, T. Moos, and A. Moradi, "Let's Take it Offline: Boosting Brute-Force Attacks on iPhone's User Authentication through SCA," in *IACR Transactions on Cryptographic Hardware and Embedded Systems - CHES 2021*, Jul. 2021.

[26] ENFSI, "Best Practice Manual for the Forensic Examination of Digital Technology," 2015. [Online]. Available: http://enfsi.eu/wp-content/uploads/2016/09/enfsi-bpm-fit-01_1.pdf

[27] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," in *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2012, pp. 521–526.

[28] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis, and Modeling," in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2013, pp. 1285–1290.

[29] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. S. Unsal, and K. Mai, "Error Analysis and Retention-Aware Error Management for NAND Flash Memory," in *Intel Technology Journal*, vol. 17, no. 1, 2013.

[30] R. Micheloni, A. Marelli, and R. Ravasio, "BCH Hardware Implementation in NAND Flash Memories," in *Error Correction Codes for Non-Volatile Memories*, 2008, ch. 9, pp. 199–247.

[31] A. Hocquenghem, "Codes Correcteurs d'Erreurs," *Chiffres*, 1959.

[32] R. Bose and D. Ray-Chaudhuri, "On a Class of Error Correcting Binary Group Codes," *Information and Control*, 1960.

[33] R. G. Gallager, *Low-Density Parity-Check Codes*. MIT Press, 1963.

[34] K. Zhao, W. Zhao, H. Sun, T. Zhang, X. Zhang, and N. Zheng, "LDPC-in-SSD: Making Advanced Error Correction Codes Work Effectively in Solid State Drives," in *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, 2013.

[35] J. Cha and S. Kang, "Data Randomization Scheme for Endurance Enhancement and Interference Mitigation of Multilevel Flash Memory Devices," *ETRI Journal*, 2013.

[36] C. Kim *et al.*, "A 21 nm High Performance 64 Gb MLC NAND Flash Memory with 400 MB/s Asynchronous Toggle DDR Interface," *IEEE Journal of Solid-State Circuits*, vol. 47, no. 4, 2012.

[37] J. P. v. Zandwijk, "A Mathematical Approach to NAND Flash-Memory Descrambling and Decoding," *Digital Investigation*, 2015.

[38] JEDEC Solid State Technology Association, "Moisture/Reflow Sensitivity Classification for Nonhermetic Solid State Surface Mount Devices," IPC/JEDEC J-STD-020D.1, 2007.

[39] IPC, "Rework, Modification and Repair of Electronic Assemblies," Tech. Rep. IPC-7711/7721, 2007.

[40] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu, "Data Retention in MLC NAND Flash Memory: Characterization, Optimization, and Recovery," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 551–563.

[41] L. Crippa and R. Micheloni, "MLC Storage," in *Inside NAND Flash Memories*. Springer Netherlands, 2010.

[42] R. Degraeve *et al.*, "Analytical Percolation Model for Predicting Anomalous Charge Loss in Flash Memories," *IEEE Transactions on Electron Devices*, vol. 51, no. 9, 2004.

[43] Y. Cai, Y. Luo, S. Ghose, E. F. Haratsch, K. Mai, and O. Mutlu, "Read Disturb Errors in MLC NAND Flash Memory: Characterization and Mitigation," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2015.

[44] Cai, Yu and others, "Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques," in *Proceedings of the International Conference on High-Performance Computer Architecture (HPCA)*, 2017.

[45] C. Friederich, "Program and Erase of NAND Memory Arrays," in *Inside NAND Flash Memories*, 2010, pp. 55–88.

[46] R. Micheloni, A. Marelli, and S. Commodaro, "NAND Overview: From Memory to Systems," in *Inside NAND Flash Memories*, 2010, pp. 19–53.

[47] Micron Technology, Inc., "How Micron SSDs Handle Unexpected Power Loss," 2015. [Online]. Available: https://www.micron.com/~/media/documents/products/white-paper/ssd_power_loss_protection_white_paper_lo.pdf

[48] K. Suh *et al.*, "A 3.3 V 32 Mb NAND Flash Memory with Incremental Step Pulse Programming Scheme," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 11, pp. 1149–1156, Nov. 1995.

[49] Y. Pan, G. Dong, and T. Zhang, "Exploiting Memory Device Wear-Out Dynamics to Improve NAND Flash Memory System Performance," in *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, 2011.

[50] H. P. Belgal, N. Righos, I. Kalastirsky, J. J. Peterson, R. Shiner, and N. Mielke, "A New Reliability Model for Post-Cycling Charge Retention of Flash Memories," in *Proceedings of the International Reliability Physics Symposium Proceedings (IRPS)*, 2002.

[51] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. Unsal, and K. Mai, "Flash Correct-and-Refresh: Retention-Aware Error Management for Increased Flash Memory Lifetime," in *Proceedings of the International Conference on Computer Design (ICCD)*, 2012.

[52] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai, "Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation," in *Proceedings of the International Conference on Computer Design (ICCD)*, 2013.

[53] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, O. Unsal, A. Cristal, and K. Mai, "Neighbor-Cell Assisted Error Correction for MLC NAND Flash Memories," in *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2014.

[54] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "Enabling Accurate and Practical Online Flash Channel Modeling for Modern MLC NAND Flash Memory," *IEEE Journal on Selected Areas in Communications*, 2016.

[55] J. H. Yoon and G. A. Tressler, "Advanced Flash Technology: Status, Scaling Trends, and Implications to Enterprise SSD Technology Enablement," in *Flash Memory Summit*, 2012.

[56] L. Zhang, Y. Tan, and Q. Zhang, "Identification of NAND Flash ECC Algorithms in Mobile Devices," *Digital Investigation*, vol. 9, no. 1, 2012.

[57] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "A Large-Scale Study of Flash Memory Failures in the Field," in *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2015.

[58] B. Schroeder, R. Lagisetty, and A. Merchant, "Flash Reliability in Production: The Expected and the Unexpected," in *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, 2016.

[59] Micron Technology, Inc., "Bad Block Management in NAND Flash Memory," 2011. [Online]. Available: https://www.micron.com/~/media/documents/products/technical-note/nand-flash/tn2959_bbm_in_nand_flash.pdf

[60] C. Zambelli, A. Chimenton, and P. Olivo, "Reliability Issues of NAND Flash Memories," in *Inside NAND Flash Memories*, 2010, pp. 89–113.

[61] E. Casey *et al.*, "Investigation Delayed Is Justice Denied: Proposals for Expediting Forensic Examinations of Digital Evidence," *Journal of Forensic Sciences*, 2009.

[62] N. Mielke, H. Belgal, A. Fazio, Q. Meng, and N. Righos, "Recovery Effects in the Distributed Cycling of Flash Memories," in *Proceedings of the International Reliability Physics Symposium (IRPS)*, 2006.

[63] K. J. Laidler, "The Development of the Arrhenius Equation," *Journal of Chemical Education*, 1984.

[64] S. Arrhenius, "Über die reaktionsgeschwindigkeit bei der inversion von rohrzucker durch säuren," *Zeitschrift für Physikalische Chemie*, 1889.

[65] N. Mielke, H. Belgal, I. Kalastirsky, P. Kalavade, A. Kurtz, Q. Meng, N. Righos, and J. Wu, "Flash EEPROM Threshold Instabilities Due to Charge Trapping During Program/Erase Cycling," *IEEE Transactions on Device and Materials Reliability*, pp. 335–344, 2004.

[66] Terasic, Inc., "Altera DE0 Board," 2013. [Online]. Available: http://de0.terasic.com/

[67] Y. Cai, E. F. Haratsch, M. McCartney, and K. Mai, "FPGA-Based Solid-State Drive Prototyping Platform," in *Proceedings of the International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2011.

[68] Cypress Semiconductor Corp., "SLC Versus MLC NAND Flash Memory," 2013. [Online]. Available: http://www.cypress.com/file/209181/download

[69] M. Kumar, "Mobile Forensics: Tools, Techniques and Approach," in *Crime Science and Digital Forensics: A Holistic View*. CRC Press, 2021, pp. 102–116.

[70] G. Gendy, "Mastering eMMC Device Programming How to Maximize Speed, Quality and Cost Savings," 2017. [Online]. Available: https://www.bpmmicro.com/wp-content/uploads/2018/12/Mastering-eMMC-Device-Programming.pdf

[71] B. Kim, D. H. Kang, C. Min, and Y. I. Eom, "Understanding implications of Trim, Discard, and Background Command for eMMC Storage Device," in *2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE)*, 2014, pp. 709–710.

[72] JEDEC, "Embedded MultiMediaCard (e·MMC) Product Standard," JEDEC Solid State Technology Association, March 2010. [Online]. Available: https://www.jedec.org/system/files/docs/JESD84-A41.pdf

[73] JEDEC, "Embedded MultiMedia Card (e·MMC) Electrical Standard (5.1)," JEDEC Solid State Technology Association, February 2015. [Online]. Available: https://www.jedec.org/system/files/docs/JESD84-B51.pdf

[74] JEDEC, "Embedded MultiMediaCard (eMMC) Mechanical Standard," JEDEC Solid State Technology Association, June 2007. [Online]. Available: https://www.jedec.org/system/files/docs/JESD84-A41.pdf

[75] A. Hoog, "Android Forensic Techniques," in *Android Forensics*, 2011, ch. 6, pp. 195–284.

[76] I. Pooters, "Full User Data Acquisition from Symbian Smart Phones," *Digital Investigation*, vol. 6, no. 3-4, pp. 125–135, 2010.

[77] G. Grispos, T. Storer, and W. B. Glisson, "A Comparison of Forensic Evidence Recovery Techniques for a Windows Mobile Smart Phone," *Digital Investigation*, vol. 8, no. 1, pp. 23–36, 2011.

[78] C. Racioppo and N. Murthy, "Android Forensics: A Case Study of the HTC Incredible Phone," in *Proceedings of Student-Faculty Research Day*, Pace University, 2012, pp. B6.1–B6.8.

[79] C. M. da Silveira, R. T. de Sousa Jr, R. de Oliveira Albuquerque, G. D. Amvame Nze, G. A. de Oliveira Júnior, A. L. Sandoval Orozco, and L. J. García Villalba, "Methodology for Forensics Data Reconstruction on Mobile Devices with Android Operating System Applying In-System Programming and Combination Firmware," *Applied Sciences*, vol. 10, no. 12, 2020.

[80] J. Lessard and G. C. Kessler, "Android Forensics: Simplifying Cell Phone Examinations," 2010.

[81] S. Beaupre, "SamDunk eMMC Backdoor Leading to Bootloader Unlock on Samsung Galaxy Devices." [Online]. Available: https://github.com/beaups/SamsungCID/blob/master/SAMDUNK_1.0-03262016.pdf

[82] J. Boukhobza and P. Olivier, "7 - Flash Translation Layer," in *Flash Memory Integration*, 2017, pp. 129–147.

[83] Open NAND Flash Interface, "Open NAND Flash Interface Specification 4.0," 2014. [Online]. Available: http://www.onfi.org/~/media/onfi/specs/onfi_4_0-gold.pdf

[84] V. Tsai, "eMMC v4.41 and v4.5 Architecture for High Speed Functions and Features," 2011. [Online]. Available: https://www.jedec.org/sites/default/files/Victo_Tsai(1).pdf

[85] Infineon Technologies AG, "X-Ray Inspection Considerations for Surface-Mounted Flash ICs," March 2021. [Online]. Available: https://www.infineon.com/dgdl/Infineon-AN98522_X-ray_Inspection_Considerations_for_Surface-Mounted_Flash_ICs-ApplicationNotes-v04_00-EN.pdf?fileId=8ac78c8c7cdc391c017d0742926a65a4

[86] H. Dogan, M. Md, N. Asadizanjani, S. Shahbazmohamadi, D. Forte, and M. Tehranipoor, "Analyzing the Impact of X-Ray Tomography on the Reliability of Integrated Circuits," in *ISTFA 2015: Conference Proceedings from the 41st International Symposium for Testing and Failure Analysis*, Nov. 2015.

[87] Netherlands Forensic Institute, "NFI Memory toolkit." [Online]. Available: https://www.forensicinstitute.nl/products-and-services/forensic-products/nfi-memory-toolkit

[88] Rusolut, "Visual NAND Reconstructor." [Online]. Available: https://rusolut.com/

[89] Soft-Center, "Flash Extractor." [Online]. Available: http://www.flash-extractor.com/

[90] N. Y. Ahn and D. H. Lee, "Forensics and Anti-Forensics of a NAND Flash Memory: From a Copy-Back Program Perspective," *IEEE Access*, vol. 9, pp. 14 130–14 137, 2021.

[91] Raspberry Pi, "Raspberry Pi 4." [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[92] L. Simon and R. Anderson, "A Security Analysis of Android Factory Resets," in *Mobile Security Technologies (MOST)*, 2015.

[93] Android Open Source Project, "Secure an Android Device," 2020. [Online]. Available: https://source.android.com/security/

[94] R. Montasari and R. Hill, "Next-Generation Digital Forensics: Challenges and Future Paradigms," in *2019 IEEE 12th International Conference on Global Security, Safety and Sustainability (ICGS3)*, 2019, pp. 205–212.

[95] X. Du, C. Hargreaves, J. Sheppard, F. Anda, A. Sayakkara, N. Le-Khac, and M. Scanlon, "SoK: Exploring the State of the Art and the Future Potential of Artificial Intelligence in Digital Forensic Investigation," in *The 15th International ARES Conference on Availability, Reliability and Security*, Aug. 2020.

[96] Blancco and Ontrack, "Privacy for Sale: Data Security Risks in the Second-Hand IT Asset Marketplace," 2019. [Online]. Available: https://www.blancco.com/resources/rs-privacy-for-sale-data-security-risks-in-the-second-hand-it-asset-marketplace/

[97] J. Schneider, I. Lautner, D. Moussa, J. Wolf, N. Scheler, F. Freiling, J. Haasnoot, H. Henseler, S. Malik, H. Morgenstern, and M. Westman, "In Search of Lost Data: A Study of Flash Sanitization Practices," in *Digital Forensics Research Conference Europe (DFRWS EU)*, 2021.

[98] JEDEC, "Universal Flash Storage (UFS) version 2.2," JEDEC Solid State Technology Association, August 2020. [Online]. Available: https://www.jedec.org/system/files/docs/JESD220C-2_2.pdf

[99] L. Cai, "Guard Your Data with the Qualcomm®Snapdragon™ Mobile Platform," Apr. 2019. [Online]. Available: https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/guard_your_data_with_the_qualcomm_snapdragon_mobile_platform2.pdf

[100] D. Giese and G. Noubir, "Amazon Echo Dot or The Reverberating Secrets of IoT Devices," in *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2021, pp. 13–24.

[101] Digi International Inc., "Secure boot flow | ConnectCore 8X," Feb. 2024. [Online]. Available: https://www.digi.com/resources/documentation/digidocs/embedded/android/dea11/cc8x/android-trustfence_r_secure-boot-flow

[102] The U-Boot development community, "Android verified boot 2.0," Apr. 2021. [Online]. Available: https://docs.u-boot.org/en/v2021.04/android/avb2.html?highlight=rpmb#avb-using-op-tee-optional

[103] Android Open Source Project, "Verifying boot," Mar. 2024. [Online]. Available: https://source.android.com/docs/security/features/verifiedboot/verified-boot#rollback-protection

[104] Google Project Zero, "Trust issues: Exploiting TrustZone TEEs," Mar. 2024. [Online]. Available: https://googleprojectzero.blogspot.com/2017/07/trust-issues-exploiting-trustzone-tees.html

[105] OMTP Limited, "OMTP advanced trusted environment TR1 v1.1," May 2009. [Online]. Available: http://www.omtp.org/OMTP_Advanced_Trusted_Environment_OMTP_TR1_v1_1.pdf

[106] ARM Limited, "ARM CoreLink TZC-400 TrustZone Address Space Controller Technical Reference Manual," 2014. [Online]. Available: https://developer.arm.com/documentation/ddi0504/c/?lang=en

[107] LineageOS, "LineageOS." [Online]. Available: https://github.com/LineageOS

[108] A. Altman and U. Hansson, "Mmc tools (mmc-utils)," 2023. [Online]. Available: https://git.kernel.org/pub/scm/utils/mmc/mmc-utils.git

[109] R. Hay, "Exploiting Qualcomm EDL Programmers (2): Storage-based Attacks Rooting," January 2018. [Online]. Available: https://alephsecurity.com/2018/01/22/qualcomm-edl-2/

[110] A. Tang, S. Sethumadhavan, and S. Stolfo, "CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management," in *26th USENIX Security Symposium (USENIX Security 17)*, Aug. 2017, pp. 1057–1074.

[111] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping Bits in Memory without Accessing Them: An Experimental Study of DRAM Disturbance Errors," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 361–372, 2014.

[112] C. Shepherd, K. Markantonakis, N. van Heijningen, D. Aboulkassimi, C. Gaine, T. Heckmann, and D. Naccache, "Physical Fault Injection and Side-Channel Attacks on Mobile Devices: A Comprehensive Analysis," *Computers & Security*, vol. 111, p. 102471, 2021.

[113] X. M. Saß, R. Mitev, and A.-R. Sadeghi, "Oops..! I Glitched It Again! How to Multi-Glitch the Glitching-Protections on ARM TrustZone-M," in *32nd USENIX Security Symposium (USENIX Security 23)*, Aug. 2023, pp. 6239–6256.

[114] P. Qiu, D. Wang, Y. Lyu, and G. Qu, "VoltJockey: Breaching TrustZone by Software-Controlled Voltage Manipulation over Multi-core Frequencies," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, p. 195–209.

[115] O. Avraham, "eMMC Hacking, or: How I Fixed Long-dead Galaxy S3 Phones," Jan 2018. [Online]. Available: https://media.ccc.de/v/34c3-8784-emmc_hacking_or_how_i_fixed_long-dead_galaxy_s3_phones

[116] oranav, "i9300_emmc_toolbox," Jan. 2024. [Online]. Available: https://github.com/oranav/i9300_emmc_toolbox

[117] NewAE Technology Inc., "ChipSHOUTER Kit," 2020. [Online]. Available: https://www.newae.com/products/nae-cw520

[118] Raspberry Pi, "Raspberry Pi Pico." [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-pico/

[119] Arm Limited, "ARMv7-M Architecture Reference Manual," Dec. 2021. [Online]. Available: https://developer.arm.com/documentation/ddi0403/latest

[120] C. Chien, "I.MX in automotive," 2017. [Online]. Available: https://www.nxp.com/docs/en/supporting-information/BL-Micro-i.MX-in-Automotive-Carl-Chien.pdf

[121] NXP, "Android Automotive OS for i.MX Applications Processors." [Online]. Available: https://www.nxp.com/design/design-center/software/embedded-software/i-mx-software/android-automotive-os-for-i-mx-applications-processors:ANDROID-AUTO

[122] NXP B.V., "i.MX Android™ Security User's Guide," May 2020. [Online]. Available: https://community.nxp.com/pwmxy87654/attachments/pwmxy87654/imx-processors/167888/2/i.MX_Android_Security_User's_Guide.pdf

[123] J. van Woudenberg and C. O'Flynn, *The Hardware Hacking Handbook*, 1st ed. San Francisco, CA: No starch press, 2022, ch. Chapter 14: Think of the Children: Countermeasures, Certifications and Goodbytes.

Acknowledgments

I would like to express my deep gratitude to my colleagues at the Netherlands Forensic Institute, especially the members of the Device Forensic team under DBS-DT. This work would not have been possible without your support. Each of you has contributed to this thesis, and every conversation served as a source of motivation throughout my Ph.D. journey.
I would also like to extend my thanks to my former colleagues at the National Police Agency of Japan, where my career as a digital forensic investigator began. During my time there, I was fortunate to have numerous opportunities to engage with diverse forensic techniques from various countries, all of which have shaped my expertise in this field.
Finally, I would like to thank my academic advisors for their invaluable input, advice, and encouragement.