D5.1

Vulnerabilities Analysis and Attack Scenarios Description

Project number:	883156
Project acronym:	EXFILES
Project title:	Extract Forensic Information for LEAs from Encrypted SmartPhones
Project Start Date:	1st July 2020
Duration:	36 months
Programme:	H2020-SU-SEC-2019, Technologies to enhance the fight against crime and terrorism
Deliverable Type:	Report
Reference Number:	SU-FCT02-883156 / D5.1 / V1.0
Work package:	WP 5
Due Date:	December 2020 – M06
Actual Submission Date:	15th January 2021
Responsible Organisation:	Royal Holloway, University of London (RHUL)
Editor:	Carlton Shepherd (RHUL)
Dissemination Level:	Public
Revision:	V1.0
Abstract:	This report presents the state of the art in physical fault injection and side-channel attacks on mobile devices. It surveys over 40 research papers from which 15 attack scenarios are sourced and compared.
Keywords:	Mobile security, fault injection, side-channel attacks

The project EXFILES has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 883156.

Editor

Carlton Shepherd (RHUL)

Contributors (ordered according to beneficiary numbers)

Carlton Shepherd (RHUL)
Konstantinos Markantonakis (RHUL)
RISCURE
CEA
IRCGN
BKA
CNI
NFI
Texplained
Cyber I
NCIS
Synacktiv

Disclaimer

Executive Summary

Mobile devices, particularly smartphones, often contain relevant forensic data critical to criminal investigations. However, investigation efforts are increasingly more complicated given the growing complexity of today's mobile platforms. Software, firmware, and hardware configurations differ significantly between manufacturers and even between devices from the same manufacturer. Furthermore, modern mobile devices contain numerous security mechanisms that hinder the extraction of forensic evidence, such as full-disk encryption (FDE), secure boot sequences, and trusted execution environments (TEEs).

Two widely studied areas for recovering device data are *fault injection* and *side-channel* attacks. These approaches can bypass conventional checks used to control access to user device data, such as password and fingerprint verification. Both attack methods focus on subverting the *implementation* of security systems, rather than weaknesses in high-level algorithm design. Fault injection attacks (FIAs) subject device components to unintended physical conditions and leverage any resulting errors to recover secret data. In contrast, physical side-channel attacks (SCAs) use side-effects produced by a device, such as electromagnetic emissions and power consumption, which can reveal secret data under execution.

This report presents an extensive survey of the state of the art in physical fault injection and side-channel attacks on mobile devices. Over 40 research publications are examined that were published between 2009 and 2020. From this survey, 15 attack scenarios are identified based on the results of this body of work, including bypassing secure boot sequences, recovering encryption keys, and gaining access to TEE-protected data. A comprehensive comparison of these works is presented alongside their prerequisites, evaluated platforms, success rate, and the attack scenarios. This information in this report is intended to provide an analysis of state-of-the-art attacks to forensic investigators for when modern mobile devices are under examination.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Digital forensics is concerned with the extraction and analysis of evidence on digital devices. Mobile devices, particularly smartphones, can contain relevant data to criminal investigations. However, data extraction and analysis methods have significantly increased in difficulty with the growing complexity of mobile platforms. Heterogeneous hardware and software configurations are commonplace, even between devices from the same manufacturer. Modern system-on-chip (SoCs), which underpin today's mobile devices, support on-board modems and wireless connectivity, graphics processing units (GPUs), and multi-core processors. SoCs also contain a variety of security extensions, including trusted execution environments (TEEs) that provide hardware-assisted isolation of applications and services, and cannot be accessed with privileged software access to the device. Full-disk encryption (FDE), hardware-assisted key management, and secure boot chains have also become ubiquitous. All of these measures have made it increasingly difficult for law enforcement agencies (LEAs) to recover forensic evidence from mobile devices.

Two widely studied approaches for data extraction are *fault injection* and *side-channel* attacks. These methods focus on leveraging weaknesses in the *implementation* of a security system, rather than in the algorithm itself. Fault injection attacks (FIAs) subject the device to physical conditions beyond that which it was intended. This includes extreme temperatures, under- or over-volting the device's supply voltage, and subjecting components to electromagnetic (EM) pulses. These can cause errors in the operation of components, which can be used to recover secret data. Physical side-channel attacks (SCAs) use side effects produced by a device while it is being executed. These side effects, such as EM emissions and power consumption, can reveal secret data if the side effect is dependent on the operation under execution. Both techniques can enable data extraction without requiring user-known keys or secrets.

In this report, the state of the art in fault injection and side-channel attacks is surveyed extensively. Over 40 papers are examined, published between 2009 and 2020, from which 15 attack scenarios are drawn. These provide a road-map as to the attacks an examiner could mount during the forensic examination of a mobile device. This includes bypassing secure boot sequences, recovering AES and RSA keys, and gaining access to TEE-resident data. A comprehensive comparison of these works is given alongside their prerequisites, evaluated platforms, success rates, and the attack scenarios to which they can be applied.

1.2 Scope

This report is concerned with physical fault injection and side-channel attacks on mobile systems. The focus is on attacks that exploit, whether wholly or in part, physical properties of the device, such as EM emissions, power consumption, and acoustic emissions. Attacks that are wholly software-based are thus outside the scope of this report. Examples of these include micro-architectural attacks that are exploitable only with software access, e.g. cache timings and side effects from speculative execution, and software timing attacks on security protocols and other systems.

1.3 Report Organisation

The following chapter (Chapter 2) presents preliminary information about modern mobile device architectures and their common security mechanisms. Chapter 3 provides a comprehensive survey of modern physical fault injection attacks, including electromagnetic fault injections (EMFIs) and voltage- and clock-based glitch attacks. Next, Chapter 4 surveys state of the art side-channel attacks, encompassing electromagnetic analysis, power analysis, and acoustic side-channels. In Chapter 5, comparison tables of state-of-the-art fault injection and side-channel attacks are presented using a common set of evaluation criteria, before discussing their broad challenges and limitations. This chapter also discusses applicable attack scenarios of each surveyed work. Lastly, Chapter 6 summarises and concludes this report.

Chapter 2

Mobile Device System Architecture

This chapter summarises the system and security architecture of modern mobile devices. It covers a high-level overview of mobile platforms and system-on-chips, before discussing mobile TEEs and their security features. It ends by examining the role of secure boot, secure key management, and full-disk encryption.

2.1 Overview

Modern mobile device systems comprise a complex and heterogeneous set of hardware, firmware and software components. This set of components can even differ between the same handset model due to geographic preferences and restrictions.

2.1.1 Hardware

While there are no strict limitations as to what a device may contain, original equipment manufacturers (OEMs) have converged on a set of high-level components that are used widely by mainstream manufacturers:

- **System-on-Chips (SoCs)**: mobile devices contain at least one SoC that contains, at the very least, the application processor (AP), RAM, ROM and persistent storage. Moreover, modern devices often delegate long-running services—sensor hubs, Bluetooth and Wi-Fi modules—to discrete children SoCs. This allows the relatively energy-intensive AP SoC to remain in a low-powered state while high-frequency, lower complexity services are fulfilled elsewhere.
- **Package-on-Packages (PoPs)**: to minimise physical footprint, OEMs can stack multiple ICs/SoCs using ball grid array (BGA). This produces single, densely packed unit that is fitted to the main PCB. The Xiaomi MI 10, for example, uses a PoP containing the main AP and a discrete RAM unit, as shown in Figure 2.1.
- **Display**: modern mobile devices contain (O)LED-based displays capable of supporting high-definition resolutions, with an integrated capacitive touchscreen for user input.
- **Baseband Processor (BP)**: older devices use dedicated baseband processor chips for supporting radio-based communications that require an antenna. Nowadays, silicon vendors integrate the BPs directly into the main SoC (see Samsung's Exynos 980 [4]). BPs

provide digital signal processing (DSP) services for sending and receiving data using common cellular protocol stacks, e.g., 4G LTE and 5G. BPs can be independent subsystems that use their own real-time operating system (RTOS) and processor.

- **Wi-Fi and Bluetooth**: similar to baseband processors, Wi-Fi and Bluetooth transceivers are often observed as independent subsystems on modern mobile devices (see Figure 2.1). These provide DSP and protocol support for the IEEE 802.11 and Bluetooth (Low Energy) specifications.
- **Near-Field Communication (NFC)**: separate transceivers that provide multi-protocol support for RFID/NFC cards, tags, and card emulation mode as per the ISO/IEC 14443, 15693, and 18092 specifications.
- **Other Peripherals**: support for a multitude of input/output (I/O) interfaces to provide connectivity with external peripherals. Common interfaces are General Purpose I/O (GPIO), UART, I2C, and SPI. Specialised peripherals found on modern mobile devices include fingerprint readers, pressure pads, proximity detectors, accelerometers, gyroscopes, and barometers.
- **External storage**: discrete memory units capable of long-term persistent storage of device data, such as embedded (eMMC) and removable memory cards.
- **External connectors**: OEMs have converged on a small number of physical connectors for communicating with external devices, namely USB (USB Mini B and USB-C) and the Apple Lightning interface.

Security Mechanisms

Specific hardware security mechanisms on modern SoCs include the following:

- **TrustZone**: a set of security extensions to the ARM architecture that allows a separate, 'secure' world to execute in parallel, outside the visibility scope of the native operating system. See §2.3.3 for further details.
- **Cryptographic co-processor**: a dedicated hardware chip for performing cryptographic operations, such as key management, digital signatures, encryption and decryption. Such processors typically employ hardware tamper resistance.
- **Tamper-resistance**: countermeasures against physical tampering of the device or a particular component, e.g., tamper-proof packaging and resistance to unusual supply voltages, clock signals, and other FIAs. Engineering debugging ports are also closed entirely, e.g., JTAG, or require authentication from an authorised engineer.

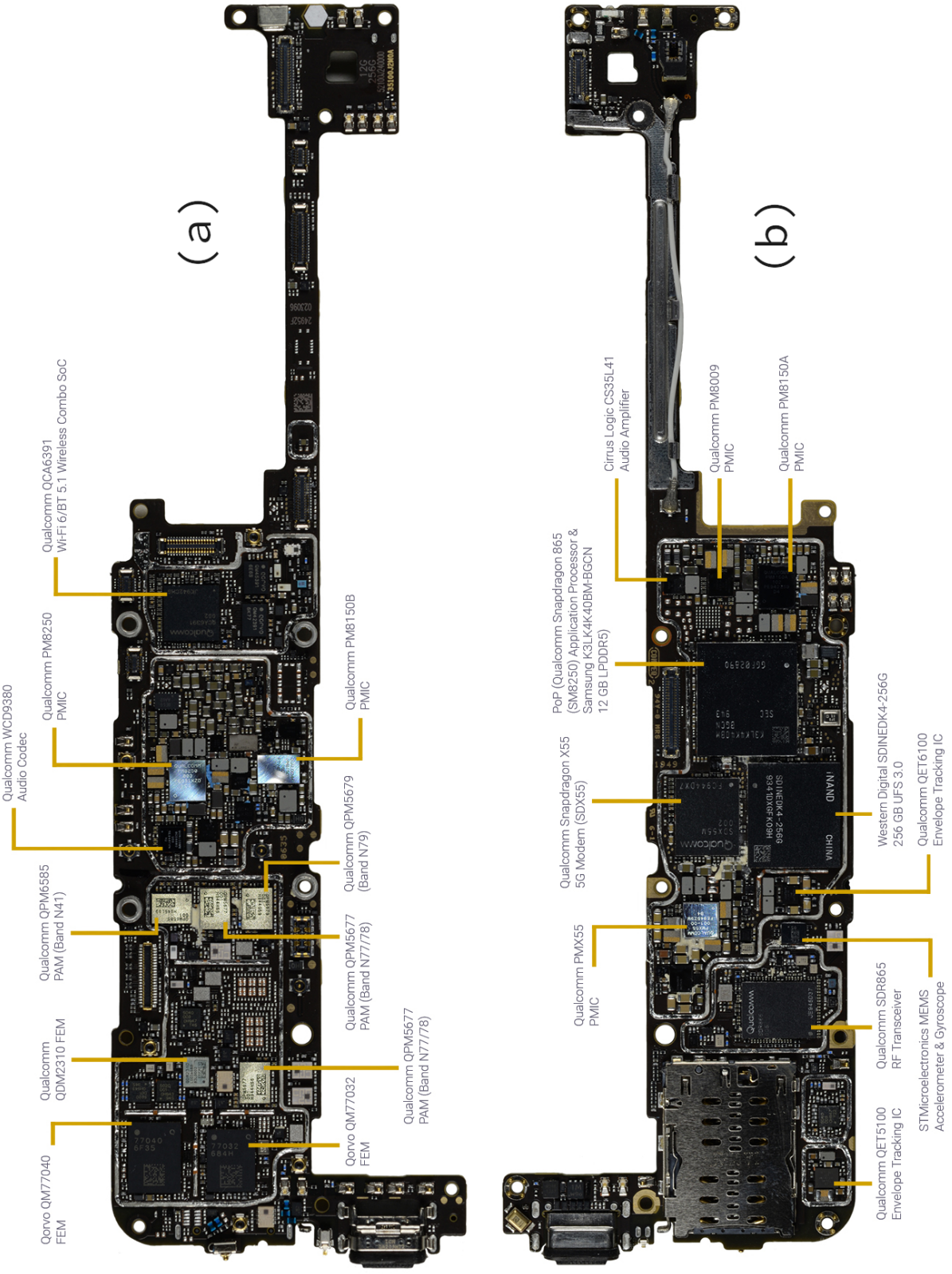

Figure 2.1: Xiaomi MI 10 main board, front (a) and rear (b), from [121].

2.1.2 Software

From a software viewpoint, mobile devices contain a dynamic set of user-space applications hosted by a fully-fledged operating system (OS). The most widely deployed of these are Apple iOS and vendor-tailored implementations of Android. An analysis of leading device platform configurations in the marketplace was provided previously in Deliverable D1.1 [44]. Mobile OSs contain a kernel, typically Linux-based, for handling memory management; inter-process communication; process scheduling; and hardware-specific drivers for audio devices, cameras, Bluetooth and Wi-Fi connectivity, and other peripherals.

Above the kernel layer, hardware abstraction layers (HALs) are used. A HAL provides a common interface to device-specific implementations of audio, camera, sensor and related peripheral drivers to the upper software layers. Native C/C++ libraries are then located at the same or subsequent layer, including the C standard library (Libc), Vulkan and OpenGL graphics libraries, and audio DSP libraries. On Android systems, the Android Runtime (ART) is located here, which translates Android application bytecode into native instructions.

The next level of abstraction is used for implementing and providing reusable high-level interfaces to user-space applications. On Android devices, this is the Java API framework that implements functions for managing notifications, activities, location and telephony services, and the view system. These functions can be utilised by third-party and OEM developers in their own applications. The highest level of abstraction is the set of user-space applications provided by the OEM or third parties. This is the set of applications that everyday users interact with: email clients, calendars, video players, games, banking applications, and so on. The Android software stack is shown in Figure 2.2 that illustrates the hierarchy of various software components.

Security Mechanisms

Dedicated software security mechanisms include the following:

- **Full-disk encryption (FDE)**: keeps users' device data encrypted at rest, requiring user authentication for it to be decrypted (see §2.4.3).
- **Trusted execution environment (TEE)**: a TEE executes in parallel with the native OS and provides security-sensitive services where the native OS cannot be trusted. On modern devices, TEEs are used to protect cryptographic keys, biometrics data, digital rights management (DRM) systems, and mobile payments (see §2.3).

2.1.3 Firmware

The final architectural component is the firmware that bridges OEM-specific hardware and the software stack. This comprises logic residing in on-SoC memory units that executes when the device is first powered on but before the main OS is loaded. It is used for performing basic power-on self-test (POST), configuring clocks and memory management units (MMUs), and initiating the boot-loading sequence. This firmware also contains vendor-specific logic for implementing a trusted execution environment. This layer is usually most opaque to security researchers: implementations vary widely between devices containing different hardware architectures, with the corresponding firmware code kept closed-source to external entities.

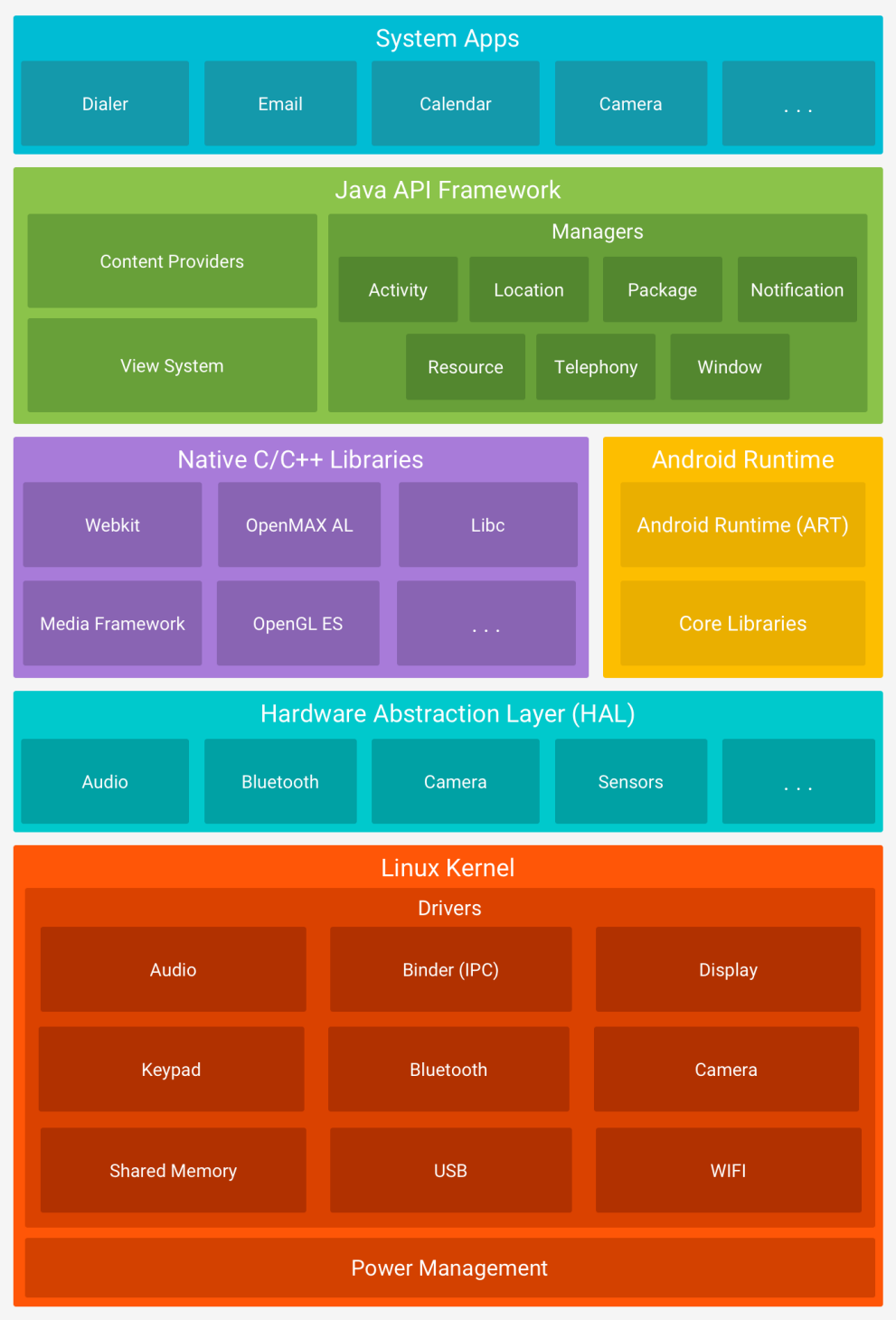

Figure 2.2: The Android OS software stack [9].

2.2 System-on-Chips (SoCs)

The centrepiece of modern mobile devices is the system-on-chip: an integrated circuit (IC) that contains the core components for establishing a working system. Mobile devices are built ubiquitously around SoCs because of their small physical footprint and greater energy efficiency versus traditional computing systems. These are designed around the configuration of reusable semiconductor intellectual property (IP) blocks. IP blocks can be developed in-house or licensed from a third party, such as ARM, and then eventually fabricated into a single IC by the SoC vendor. A variety of IP blocks exist on the marketplace for providing application processors [18]; security extensions and cryptographic accelerators, e.g., the ARM TrustZone and CryptoCell families [19]; and graphics and multimedia units, such as the ARM Mali series [17].

As such, today's SoCs comprise a multitude of subsystems. High-frequency multi-core central processing units (CPUs) with caches and memory management units are at the centre of modern SoC design. Around this, dedicated memory units are commonplace, such as dynamic and static RAM, ROM and flash memory. Other common subsystems are timers and clocks and cryptographic co-processors for accelerating cryptographic operations, including digital signatures and symmetric and asymmetric encryption and decryption. Integrated GPUs and audio DSP units are also found widely for video and audio processing respectively. All of these components communicate with one another via on-SoC system buses that control and perform data transfer between subsystems.

The precise features, performance and capabilities of a SoC varies between vendors based on their preferences. At the time of writing, Qualcomm, MediaTek, Texas Instruments, Huawei, and Samsung are among the most popular SoC vendors in the commercial marketplace. The reader is referred to Table 9 in D1.1 for a comprehensive list of SoCs used by modern mobile devices [44].

2.3 Mobile Trusted Execution Environments (TEEs)

Trusted execution environments (TEEs) aim to defend security-sensitive assets from privileged software attacks from the native OS. On mobile devices, TEEs protect cryptographic keys and biometric authentication, payment processing, and digital rights management algorithms. TEEs execute in parallel with the native OS and its application using hardware-assisted access control on the device's system-on-chip. While TEEs rely on a hardware-based implementation, they do not provide protection against a wide range of hardware attacks [74]. An array of TEE systems and paradigms have been developed by industry and academia. Today, the most common commercial TEEs are Intel Software Guard Extensions (SGX) and ARM TrustZone. However, it is ARM TrustZone and the GlobalPlatform TEE—a set of specifications relating to TEEs—that pervade today's mobile devices [14, 42, 118]. We discuss these technologies and their security properties next.

2.3.1 High-Level Architecture

The GlobalPlatform TEE is the prevailing set of specifications for governing the architecture, management and security goals of TEEs [51, 54, 55]. It establishes two worlds of execution: the rich

execution environment (REE)[1] and the trusted execution environment (TEE)[2]. The GP TEE architecture is depicted in Figure 2.3. The REE contains the mobile's native OS, such as Android, its firmware and user-space applications. In contrast, the TEE contains an independent security kernel, known as the trusted OS, and a set of security-sensitive trusted applications (TAs). The REE communicates with the TEE via a tightly controlled interface at the platform's highest privilege level, known as the secure monitor.

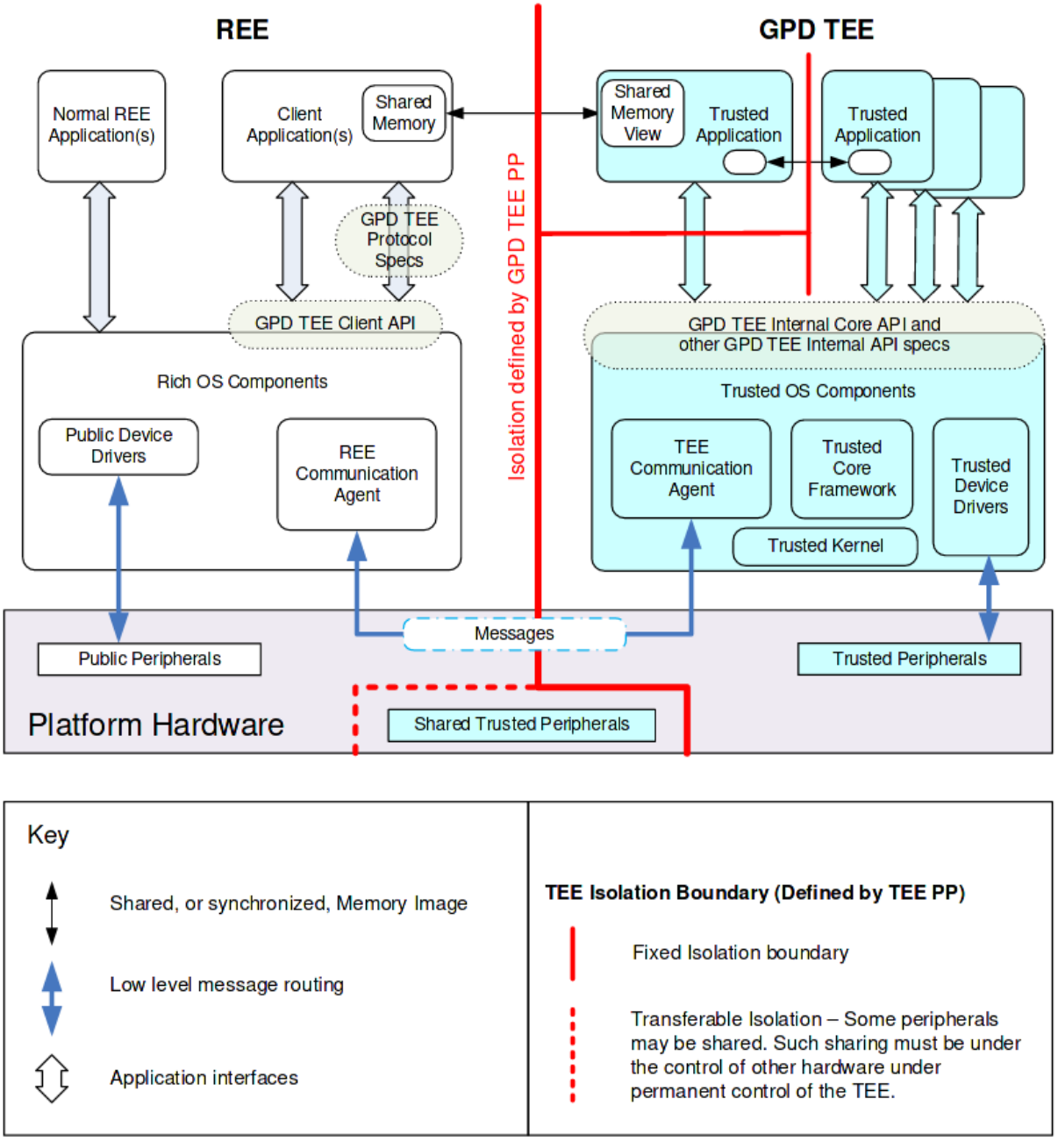

Figure 2.3: High-level GP TEE system architecture [51].

The secure and non-secure worlds co-exist on the same underlying platform hardware, which execute concurrently using hardware-assisted isolation. This isolation is critical to prevent REE

[1] Also called the *untrusted* or *non-secure* world.
[2] Also called the *trusted* or *secure* world.

adversaries from accessing secure world memory regions and peripheral devices. Upon launch, the GP TEE must be initialised from a root-of-trust (RoT) using a secure boot process. This process is described further in §2.4.1 for ARM TrustZone, which is the means by which GlobalPlatform TEEs are instantiated on ARM-based mobile devices. Ultimately, this initialisation process must be performed without trusting or co-operating with the untrusted REE.

The GlobalPlatform TEE specifies the GlobalPlatform Internal API [52] by which trusted world applications (TAs) communicate with the trusted OS. GlobalPlatform also specifies the Client API [50], which standardises the communication interfaces between REE applications and the TAs via the non-secure and secure world OSs. For the full documents, the reader is referred to the GlobalPlatform Technology Document Library [53], where the specifications are publicly available.

2.3.2 GlobalPlatform TEE Threat Model

The GP TEE Protection Profile (GP TEE PP) is used for a GP TEE target of evaluation (TOE) under the Common Criteria Framework. The GP TEE PP *"targets threats to the TEE assets that arise during the end-usage phase and can be achieved by software means...focuses on non destructive software attacks that can be easily widespread...and constitute a privileged vector for getting undue access to TEE assets without damaging the device itself"* [55].

In broad terms, the secure monitor, TEE kernel and TAs are considered trusted. Errors in any of these components, such as the functions it exposes to non-secure client applications and world context switching logic, can compromise the intended services that a TEE aims to provide. With respect to hardware, the GP TEE can have access to a secure clock, cryptographic accelerators, and volatile and non-volatile memory. However, the TEE must access these resources without cooperating with the untrusted REE; the GP TEE must be self-sufficient from any software or firmware components in the untrusted REE [51, 55].

The GP TEE PP provides a series of assets and the security properties that should be satisfied, which is reproduced in Table 2.1. It covers software and hardware assets used by the TEE and its TAs, including TEE firmware, sources of time, run-time and persistent memory sources, and TEE keys. The defined security requirements comprises, among others, the confidentiality, integrity, authenticity, monotonicity and immutability of particular TEE assets. These assets should be protected, as a minimum requirement, against two high-level REE adversary types defined in the GP TEE PP:

- **Basic remote attacker**: *"Performs the attack on a remotely-controlled device or alternatively makes a downloadable tool that is very convenient to end-users. The attacker retrieves details of the vulnerability identified in the identification phase and [...] makes a remote tool or malware and uses techniques such as phishing to have it downloaded and executed by a victim [into the untrusted world]."*
- **Basic on-device attacker**: *"Has physical access to the target device; it is the end-user or someone on his behalf. The attacker is able to retrieve exploit code, guidelines written on the internet on how to perform the attack, and downloads and uses tools to jailbreak/root/reflash the device in order to get privileged access to the REE allowing the execution of the exploit. The attacker may be a layman or have some level of expertise but the attacks do not require any specific equipment."* [55].

It is noted that the minimum assurance level for GP TEE compliance is CC EAL2. This is lower than other secure execution platforms, such as smart cards and secure elements. These platforms are usually evaluated to CC EAL4+ and against expert adversaries possessing specialist testing equipment, which the TEE is not designed to defend against.

Table 2.1: Minimum security requirements of GP TEE assets, from [55, 114].

	Property									
Asset	**C**	**I**	**AU**	**U**	**UP**	**AT**	**DB**	**M**	**CO**	**IM**
TEE Identifier				✓						✓
RNG					✓					
TA Code			✓						✓	
TA Data and Keys	✓	✓	✓			✓	✓		✓	
TA Instance Time								✓		
TA Run-time Data	✓	✓							✓	
TA Persistent Data	✓	✓	✓				✓		✓	
TEE Firmware		✓	✓							
TEE Init. Code and Data		✓								
TEE Storage RoT	✓	✓								
TA Persistent Time								✓		
Rollback Detection Data		✓								
TEE Debug Auth. Key	✓	✓								

C: Confidentiality, **I**: Integrity, **AU**: Authenticity, **U**: Uniqueness, **UP**: Unpredictability, **AT**: Atomicity, **DB**: Device Binding, **M**: Monotonicity, **CO**: Consistency, **IM**: Immutability.

2.3.3 ARM TrustZone

ARM TrustZone is a set of extensions to the ARM processor architecture that provides a secure world of execution; it is the main way by which the GlobalPlatform TEE is realised on ARM-based platforms [14, 42, 118]. Like the GlobalPlatform TEE, TrustZone divides execution into 'secure' and 'non-secure' worlds that host security-sensitive and non-sensitive material respectively. Each world can host its own independent OS and set of applications that execute at varying exception levels aboard the device, which is illustrated in Figure 2.4. The secure world OS is a specialist security kernel with limited functions to reduce the scope for vulnerabilities. The secure world OS implements the respective GlobalPlatform TEE APIs, e.g. Internal API [52], and implements memory management and process scheduling of TEE applications. Some commercially available secure world OSs are Huawei's iTrustee [64], Samsung's TEEGRIS [107], Trustonic's Kinibi [126], and the Qualcomm TEE [101].

At run time, ARM TrustZone uses a control signal called the 'non-secure' (NS) bit to reflect the world of execution. This is the main mechanism for informing the security status across the device. The NS bit is propagated through all areas of the SoC where non-secure world adversaries may attempt to access secure world material. This includes page tables, cache lines, and bus transactions to memory firewalls and peripheral controllers. Any unauthorised non-secure access attempts to secure world-only resources are prevented by the corresponding on-SoC controller. Such controllers are the TrustZone Peripheral Controller (TZPC) for protecting secure world-only

peripheral interfaces, and the TrustZone Application Space Controller (TZASC) for partitioning dynamic memory regions. Any SoC bus transactions that carry a non-zero NS bit are blocked by these controllers depending on their configuration.

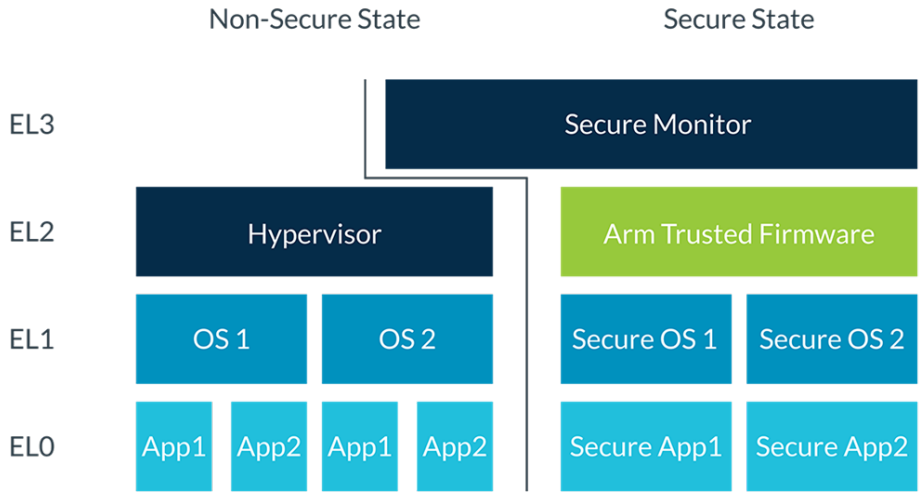

Figure 2.4: ARM exception model for v8.4-A and beyond [15].

To enable inter-world communication, messages are transmitted over a secure monitor that executes at the highest exception level on the ARM architecture (EL3). User-space applications communicate with the TEE through a library by the trusted OS provider, which communicates with a driver deployed in the OS kernel. This driver, also provided by the trusted OS provider, uses the secure monitor call (SMC) interface to enter into TrustZone's secure monitor mode that securely performs the world context switch. Data is passed to the secure world kernel and then to a target trusted application. Responses are returned using the reverse process. At boot time, ARM TrustZone uses a secure boot process, discussed in §2.4.1 to securely authenticate and initialise the secure monitor firmware and for loading the TEE system image. A reference implementation of the secure monitor code and bootloading process is provided as part of the Trusted Firmware project [78], which OEMs can tailor to their own platform configurations.

2.3.4 Proprietary Systems

Proprietary systems have been developed by manufacturers that offer orthogonal services to the GlobalPlatform TEE. Three common systems found on Samsung, Apple and Google devices are Samsung KNOX, Apple Secure Enclave Processor (SEP), and Google Titan M respectively.

Samsung KNOX is a security platform found on leading Samsung handsets built from a TrustZone-based TEE. The platform provides containers, known as KNOX Workspaces, that are encrypted using keys held in the TEE. The workspaces provide secure locations in which documents, videos, photographs and other data can be stored and encrypted automatically. As such, data cannot be extracted from these workspaces without co-operation from the TEE to decrypt them. At boot time, Samsung KNOX extends the secure boot process on Samsung devices to set one-time programmable ROM (eFuse) if unauthorised boot components are loaded. If the eFuse is set, new KNOX containers cannot be created and existing ones cannot be decrypted. The state

of this eFuse is permanent and persists across reboots and resets [41]. Another additional security mechanism is the use of remote attestation to allow remote entities to validate the state of the target device. That is, the remote verifier can learn, using a secure channel, if the device has been booted with any unauthorised bootloaders by inspecting the state of the eFuse. These attestation messages are signed using a device-specific attestation key accessible only to the TEE, the public key of which is used by the remote verifier to authenticate the response messages [41].

Apple's Secure Enclave Processor (SEP) is a security co-processor on Apple devices. Very few publicly available details exist about the SEP besides high-level features provided in Apple marketing materials. However, some reverse engineering methods have been made by independent security researchers [81]. The SEP shares similarities with ARM TrustZone and is directly initialised during the secure boot process by the boot ROM, the device's root-of-trust. It provides access to hardware-backed cryptographic operations, such as encryption and signing services. The SEP uses memory encryption between internal memory and external RAM to protect data in transit between itself and and main memory. It can also access its own set of peripherals, a bank of eFuses, and widely used input/output interfaces, such as GPIO and SPI. The SEP has been shown to support iOS key management and the Apple iOS biometric authentication systems, including Touch ID and Face ID.

In 2017, Google announced the Titan M tamper-resistant security co-processor for the Pixel 2 smartphone that executes independently from the device's main SoC [58]. The chip contains its own flash memory, CPU, and RAM in a single package, which is resistant to physical penetration and side-channel attacks including power and electromagnetic analysis. It is also resilient against FIAs, such as voltage, clock and temperature faults. The Titan M stores the last known safe Android version and prevents rollback attacks during the verified boot process on Pixel 2 and 3 devices. It also prevents attackers on the Android system attempting to unlock the bootloader. Moreover, the Titan M is used for lock screen verification in order to deactivate full-disk encryption and allows applications to generate and store keys via the StrongBox KeyStore APIs.

2.4 Common Mobile Platform Security Mechanisms

2.4.1 Secure Boot

Securely booting and initialising a TEE is critical to maintaining its security; the ability to load untrusted bootloaders and TEE system images would compromise the security that it intends to provide. Modern devices contain multiple bootloader stages of increasingly complexity. After receiving a reset signal, the first stage begins with performing basic SoC setup operations, including power-on self-test (POST) and clock initialisation. After this, the trusted world and secure monitor firmware is loaded, which is executed *before* the loading the native operating system. This ensures that the loading of the trusted world is not dependent on any non-secure world components. In the final stage, the boot sequence loads the non-secure world boot image that loads the operating system and user applications. An overview of the ARM boot procedure is shown in Figure 2.5.

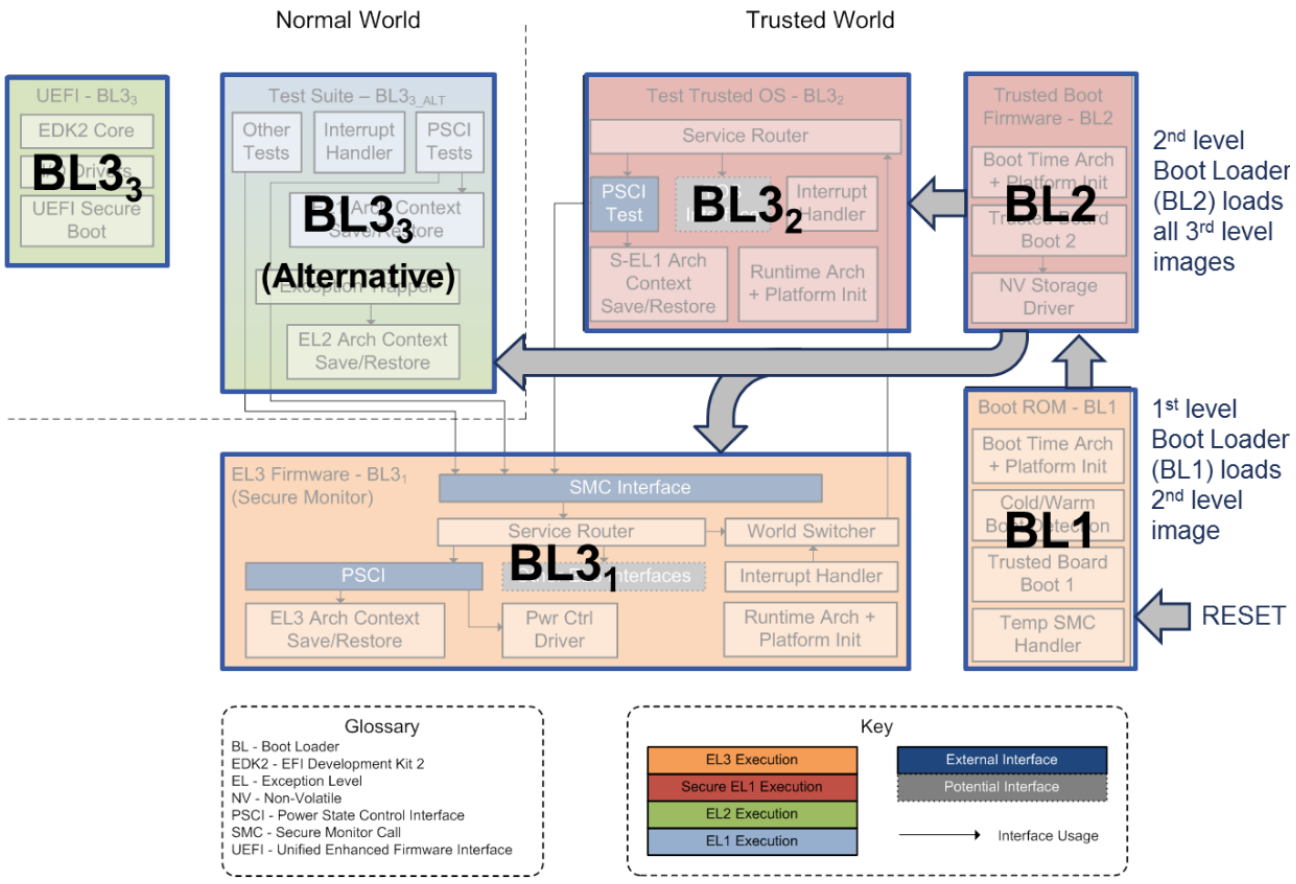

Figure 2.5: The ARM boot procedure [16].

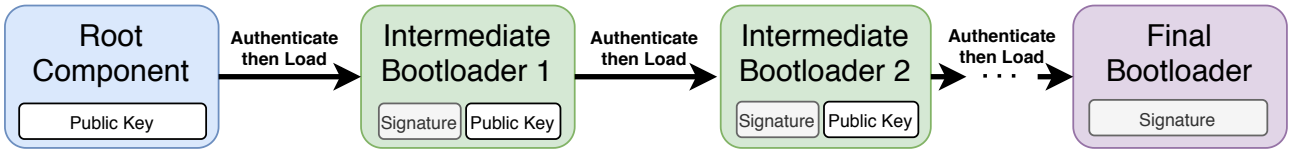

Figure 2.6: Authenticated secure boot process.

Secure boot is the process by which this is performed in an authenticated fashion. Common practice is to establish a sequentially verified chain beginning with a root of trust (RoT). This is a small, typically hardware-bound and read-only trusted component like on-SoC ROM. The RoT has secure access to a read-only public key certificate, which is used to verify the signature of the next bootloader in the chain. Upon successful signature verification, the next bootloader is loaded and control is passed to it. This next bootloader, which itself contains a public key, verifies the following bootloader and loads it. This process repeats until the final bootloader is successfully verified after which it loads the rest of the system. Bootloaders are signed using the OEM's private key; if signature verification fails—for example, after the loading of an unauthorised component—then the boot process stops. This process repeats if the device is restarted while the unauthorised component is still present. As mentioned previously, some device platforms contain one-time programmable (OTP) fuses that are permanently 'blown' if signature verification fails during the boot process [108]. The secure boot process is illustrated in Figure 2.6.

Public information is scarce regarding the specific algorithms used by OEMs for secure boot. However, we note that the ARM Trusted Board Boot requirements, which specify the boot pro-

cess for application processors designed to be compliant with the GlobalPlatform TEE Protection Profile, specifies the use of SHA-256 and RSA-2048 or 256-bit ECDSA for bootloader certificates, and AES-128 for hardware-based keys [16].

2.4.2 Secure Storage and Key Management

Mobile TEEs can provide secure persistent storage of sensitive material. At a specification level, the GlobalPlatform TEE provides functions via the Internal Core API [52] for data 'binding', which generates encryption keys from a hardware unique key (HUK) using a key derivation function (KDF) within the TEE. These keys are used to encrypt TA and TEE key material and data; TAs can also use these keys to encrypt non-secure world data. The keys must never leave the TEE and, consequently, encrypted material cannot be decrypted on another device. The GlobalPlatform TEE specifications also supports the storage of keys and other data to external security hardware, such as a secure element. When used, this external hardware must be accessible only to the TEE [51]. The GlobalPlatform TEE supports a large range of algorithms that an OEM may use for internal TEE cryptographic operations. This includes 128/192/256-bit AES in various modes of operation, e.g., CBC, XTS, and CTR; up to 2048-bit RSA; 160- to 521-bit ECC, including ECDSA and ECDH; and 128- or 196-bit triple DES. The reader is referred to the GlobalPlatform Internal Core API [52] for a full list of supported cryptographic algorithms.

An example TEE key management system is the Android Keystore, which lets (user-space) application developers generate and containerise cryptographic keys within a TEE [8]. The key material is generated using a TEE-based secret and never exposed outside the secure world, even if the OS is compromised. Android OS provides abstract interfaces to developers in order to operate upon this material, such as encryption, decryption, and signature verification and signing, but direct access to these keys is not given to developers. Related to this, the Android Gatekeeper [7] performs user authentication of inputted passwords in the TEE. The Gatekeeper enrolls user passwords by applying an HMAC with the password, identifier and a hardware secret key, which is stored in the TEE. Subsequent authentication attempts involve regenerating this value and checking equivalence. If successful, the Gatekeeper uses a TEE-derived secret to sign and transmit an authentication message to the Keystore TA. This notifies the Keystore TA that the user authentication secret was entered correctly and TEE-resident keys can be used. The Gatekeeper service limits the number of failed verification attempts and prevents further attempts using a timeout and counter.

On Apple devices, the Secure Enclave Processor (SEP, see §2.3.4) implements key management for encrypting data at rest, secure boot in macOS, and biometrics. Each SEP is built with a hardware-based unique ID (UID) from which cryptographic keys are derived, thus providing device-binding and preventing attacks that involve memory device transplantation. Additional class keys are derived from a user-inputted password and a hardware secret, which are used for services such as per-file and volume encryption. Encryption keys are never directly exposed to the device CPU. Moreover, the Apple T2 security chip, which contains the SEP, has a dedicated AES-256 engine that is built into the direct memory access (DMA) path between the flash storage and main system memory. This is used for fast file encryption from the naive OS using keys from the SEP. The reader is referred to [12] and [11] for Apple-specific security guidance.

2.4.3 Full Disk Encryption

Full disk encryption (FDE) is used to encrypt all device user data such that it cannot be readable by an attacker who has not decrypted its contents. FDE implementations use symmetric encryption, usually AES, to encrypt data at the block level. User data is encrypted before saving to it to disk, such as a photograph or audio file; when the data is read, the decryption procedure is performed before returning it to the parent process. Android's FDE implementation operates as follows: a master key (up to 256 bits) is used to encrypt/decrypt data, which is generated and hashed with a default password and a salt value when the device is booted for the first time. This hash is signed by the TEE, and the signature of this hash is used to encrypt the master key. When the user sets their PIN or password, the 128-bit key is re-encrypted and stored. Only with successful user authentication is the FDE key released such that it can be used to decrypt the contents of user data. Without successful authentication, the OS is not able to decrypt and read the data [6].

2.4.4 File-based Encryption

File-based encryption (FBE) refers to the encryption of data at a filesystem level, rather than a block or volume level as with FDE. This can allow different user profiles to exist under separate security policies; for example, a dedicated work profile that encrypts corporate files. FBE typically uses AES on commercial devices. On Apple devices, a 256-bit per-file encryption key (FEK) is generated each time a file on the data volume is created by Apple's proprietary Data Protection module, which uses the SEP for key management. The FEK is wrapped using a class key (see §2.3.4) and given to the hardware AES engine, which encrypts the file under the FEK as it is written to flash memory. When a file is read, it is decrypted by the AES engine as it is transmitted from memory. The encryption uses AES-128 in XTS mode, using 128 bits of the 256-bit key as the cipher key and the other 128 bits as the tweak.

FBE is also supported by Qualcomm-based SoCs (Snapdragon 855 and over) [100]. Here, FBE Credential Encrypted (CE) class keys are generated that are protected by a secret derived from a device unique key and a synthetic password. The synthetic password is generated by Android for each user, which is protected by the user credential, e.g. password, and (if applicable) an escrow token for providing access to system administrators. When the device is unlocked, the FBE CE key is decrypted and set in the Linux kernel key-ring. When the file system driver reads or writes a protected file, the key is retrieved from the kernel key-ring and set in the proprietary Qualcomm Inline Crypto Engine (ICE) for encrypting/decrypting files to and from memory using AES [102].

Chapter 3

Fault Injection Attacks

In this chapter, the state-of-the-art of fault injection attacks is explored. It focuses on published works that have successfully attacked mobile devices or their constituent components as a target of evaluation.

3.1 Overview

Fault injection attacks (FIAs) are active attacks that physically perturb the device beyond its intended operating conditions. The goal is to generate and observe the results of abnormal system behaviour that an attacker can leverage to access restricted functionality and data. FIAs have a long-standing, approximately 25 year history, and have been analysed in great depth in the wider smart card literature [27, 5, 34, 133, 23, 94, 61]. FIAs are usually categorised as *transient* or *permanent*, and *invasive* or *non-invasive*. Transient faults are temporary errors that the system eventually recovers from following a reset or when the fault source stops. Their aim is to temporarily disrupt the system's control flow or corrupt the results of an instruction to gain access to unauthorised functionality and data. Permanent faults indefinitely change the target component's state, which persists through device restarts and resets.

Invasive FIAs involve significant tampering to the device's internal circuity. This includes depackaging the SoC or IC under evaluation, removing protective layers, and directly inducing faults into internal components. Light pulses using high-energy light sources, near-infrared lasers, and ion beam attacks fall under this category. These techniques have been used to flip the state of transistors in memory cells and other components using carefully targeted beams [119, 127, 97, 109, 125, 75, 128]. Notably, Vasselle et al. [128] demonstrated a laser-based FIA to bypass the secure boot process of an undisclosed Android smartphone with an ARM Cortex A9-based SoC. This was achieved through physical and software-based reverse engineering of the SoC, and decapsulation of the package. This leads to a major challenge of invasive FIAs: there is a significant risk that components can be irreparably damaged, which can lead to the irreversible loss of forensic evidence when a device is under examination.

We also note that attacks using visible light and UV have become less practical as transistor gate lengths have shrank from advances in semiconductor fabrication methods. The costliest attacks employ focused ion beams (FIB) and heavy-ion microbeams (HIMs), which operate with a high degree of precision (2.5nm). Works by Li et al. [75] and Torrance et al. [125] have demonstrated attacks against cryptographic systems using HIB and FIB respectively. Generally, FIB and HIM attacks are extremely expensive ($3,000–$100,000+ USD) and require access to high levels of

expertise, specialist laboratories and testing equipment.

In contrast, non-invasive FIAs require little-to-no tampering and their effects usually disappear when the stimulus is removed or the device is reset. Non-invasive FIAs are performed by manipulating physically accessible components to generate internal faults. Well-studied non-invasive FIAs are voltage-based glitch attacks, which under- or over-volt the device's intended supply voltage [98, 26, 123, 122, 124, 25, 90]; clock glitch attacks that under- or over-clock the device to trigger faults [33, 2, 120, 71]; and exposing components to strong, targeted electromagnetic disturbances [40, 103, 43] and extreme temperatures [70, 72, 59]. In general, non-invasive FIAs are less costly, require cheaper and non-specialised equipment, and risk significantly less damage to the target component than invasive FIAs [27].

A summary of FIAs, their relative complexity, precision and risk is provided in Table 3.1. In the next sections, the state of the art in non-invasive physical FIAs—voltage-based glitches, clock glitches, electromagnetic fault injections (EMFIs), and temperature attacks—is examined extensively.

Table 3.1: Summary of FIAs based on [27] and [75].

Technique	Precision (Space)	Precision (Time)	Cost	Skill	Risk
Clock Glitch	Low	High	Low	Moderate	Low
Voltage Glitch	Low	Moderate	Low	Moderate	Low
Heating	Low	Low	Low	Low	Moderate
EMFI	Moderate	Moderate	Low	Moderate	Moderate
Light Pulse	Moderate	Moderate	Moderate	High	Moderate
Laser Beam	High	High	High	High	High
FIB	Very High	Very High	Very High	Very High	High
HIM	Very High	Very High	Highest	High	High

EMFI: Electromagnetic FI, **FIB**: Focussed ion beam, **HIM**: Heavy-ion micro-beam.

3.2 Voltage-based Glitch Attacks

Voltage-based glitch attacks manipulate the target device's supply voltage. By under- or over-volting this source, or redirecting it to ground to generate brownouts, an attacker can generate single- or multi-bit faults during execution. Consequently, the processor can be coerced into corrupting or skipping instructions, including security-critical verification checks. These attacks have been used to recover cryptographic key material, skip signature verification steps, and bypass system access control measures. The state of the art in these attacks is now described.

In 2009, Barenghi et al. [25] presented a voltage glitch attack against a software RSA implementation on a 32-bit ARM9 microprocessor (ARM926EJ-S, released in 2001). The attack exploits the relatively intensive nature of load instructions, which were found to be susceptible to mis-executions when the device was provided a low supply voltage (underfeeding). During the instruction fetch phase, the authors were able to induce faults to change the binary encoding of logical operations (`AND` to `EOR`), conditional additions (`ADDNE` to `ADDEQ`), and conditional branches (`BNE` to `BEQ`). Using this, three attacks on OpenSSL 0.9.1i were presented: (1), factoring the RSA modulus when using the Chinese Remainder Theorem (CRT); (2), an e-th root attack to retrieve an input message encrypted under a correct and faulted encryption; and (3), a theoretical attack for recovering the secret key during message signing. The instruction swaps occur only a small

number of times and *"may be reduced up to a single one in the whole computation of a target algorithm"*. Due to this rate, the key recovery attack was not practical. For the first attack, 6.6%–39% RSA-CRT computations were faulted of which 3.0%–39% were exploitable. For the second, 36.42%–62.77% of 1000 faulted runs were exploitable.

Later, Barenghi et al. [26] used the same approach—exploiting corrupted `LOAD` instructions—to recover round sub-keys in the AES cipher. The attack is independent of the key length or the number of rounds, and requires a fault-free ciphertext and a small number of faulty ciphertexts generated from the same plaintext. The attack is demonstrated using the same hardware as [25] using Linux 2.6 and an AES implementation based on OpenSSL. The authors show it is possible to break AES using 100,000 encryptions with different plaintexts and 2,000,000 encryptions of the same plaintext.

In 2016, Timmers et al. [122] presented two attacks that use a voltage-based glitch to alter the program counter (PC) register of an ARM Cortex-A9 AP on a Xilinx Zynq-7010 SoC. It uses an instruction corruption exploit with two load instructions (`LDR` and `LDMIA`) when the device is under-volted, which alters the PC on 32-bit ARM architectures (AArch32). The first attack targets the device's first bootloader—BL1, see §2.4.1—to bypass the secure boot process. The attack requires the BL1 image to be overwritten with a malicious payload containing shellcode and pointers. The fault must be injected precisely after the device has copied the shellcode into volatile memory and while it is copying the pointers. The fault, if successful, corrupts the instruction so the pointer is copied into the PC register, thus executing the shellcode and transferring control to the attacker. The second attack uses the same approach to compromise the TEE after it has booted. It exploits the scenario where the REE copies data to the TEE via a shared buffer. The attacker loads the shellcode into the buffer and induces the fault when the world switch occurs. The fault overwrites the PC with a pointer to execute the shellcode, therefore compromising the secure world. 10,000 fault attacks on the `LDR` and `LDMIA` instructions were conducted. For `LDR`, only a single (0.01%) glitch was successful; for `LDMIA`, 27 glitches were successful (0.27%). We note that attacks on actual TEE or secure boot process on a consumer device were not shown.

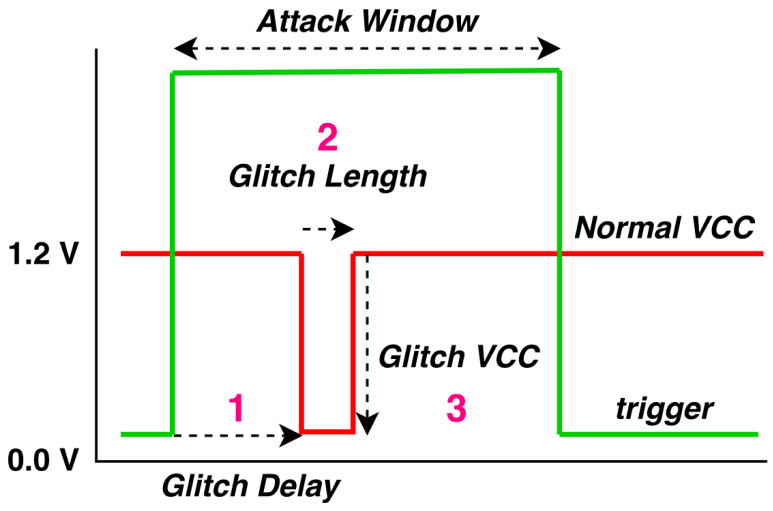

Figure 3.1: Voltage-based FI glitch parameters [123].

In 2017, Timmers and Mune [123] described three voltage-based FIAs that escalate the privilege of user-space Linux applications. The attacks assume physical access to the device and the

ability to run arbitrary Linux applications in an unprivileged context. The glitches are performed after carefully characterising the glitch parameters: normal voltage, glitch voltage, glitch length, and glitch delay (see Figure 3.1). Attack one uses an FIA during the `open` syscall when opening `/dev/mem/` to allow an unprivileged application to map arbitrary addresses into physical memory. This is performed using an instrumented untrusted application and injecting the fault during the check performed by the Linux kernel, which causes it to be bypassed. Attack two uses the same instruction corruption attack as [122] to change the PC register within the Linux kernel, triggering system crashes as a proof-of-concept. The third attack targets the `setresuid` syscall to set the unprivileged application's process ID to root. This FIA is mounted during the kernel's verification process, causing it to be bypassed. The rate of successfully injecting faults was 0.53% (attack one), 0.63% (attack two), and 0.41% (attack three).

In 2019, Qiu et al. [98] presented the VoltJockey attack on TrustZone-based systems. VoltJockey exploits a software-based vulnerability in the SoC's dynamic voltage and frequency scaling (DVFS) framework. DVFS regulates the frequency or operating voltage of a multi-core processor to minimise its power consumption; it uses kernel drivers to control the device's power management integrated circuit (PMIC) through software. The authors show that this functionality can be carefully manipulated to lower the voltage to generate cross-core faults; the attack path is shown in Figure 3.2. The attack assumes root access, a multi-core processor, and that the voltage of the target multi-core processor is software-modifiable. Because TrustZone TEEs use the same physical processor as the non-secure world, the authors were able to leverage faults mounted in the non-secure world to recover an AES key in the secure world using a reference AES implementation. This is caused by inducing byte errors in the eighth-round of the AES algorithm while it is being executed in the secure world. Attacks on RSA signature verification are also demonstrated using a reference RSA implementation. Here, the authors show that desired plaintexts can pass signature verification after introducing byte faults into the RSA public modulus. This was exploited to enable the loading of unsigned TEE applications by the TrustZone secure OS. The attacks are conducted on the Google Nexus 6 smartphone with a Qualcomm APQ8084AB SoC. A success rate of up to 2.2% for AES and 4.6% for RSA is shown. For non-mobile systems, similar attacks have been mounted subsequently against the Intel SGX TEE for the X86-64 architecture [38, 99].

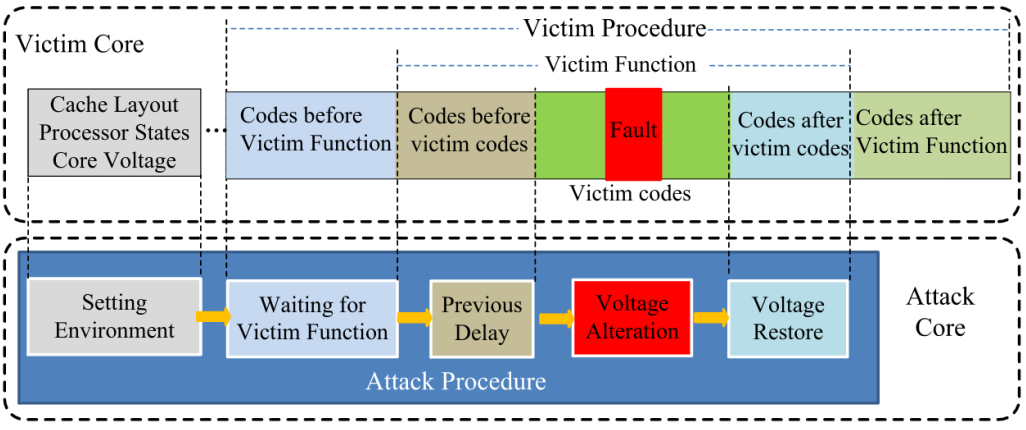

Figure 3.2: VoltJockey attack sequence [98].

In 2020, the NCC Group published a voltage-based glitch vulnerability on a MediaTek MT8163V SoC (64-bit ARM Cortex-A53) [92]. These SoCs contain a one-time programmable eFuse bank and read-only boot ROM. This acts as the root of trust in the secure boot chain (see §2.4.1) and

for storing the initial certificate used for bootloader signature verification. The authors discover, and successfully apply, a voltage glitch after the first bootloader is loaded from eMMC into RAM, which skipped the secure boot signature verification check. This leads the authors to insert and load of an unauthorised boot component that is able to perform arbitrary code execution. The success rate of the voltage glitch was 15.21%–23.44%. While not demonstrated, the authors posit that the vulnerability could be exploited to load an unauthorised TrustZone TEE image.

3.3 Clock Glitch Attacks

Clock glitches manipulate external device clocks, such as introducing additional clock edges (see Figure 3.3), that can result in instruction misses and data misreads. Instruction misses are caused by the circuit executing instruction before the processor has completed executing the previous instruction. Similarly, data misreads are caused by circuits attempting to read a value from a data bus before the memory latches out the requested value in time. Historically, clock glitches have attracted significant attention in the smart card and secure element literature [22, 82, 83].

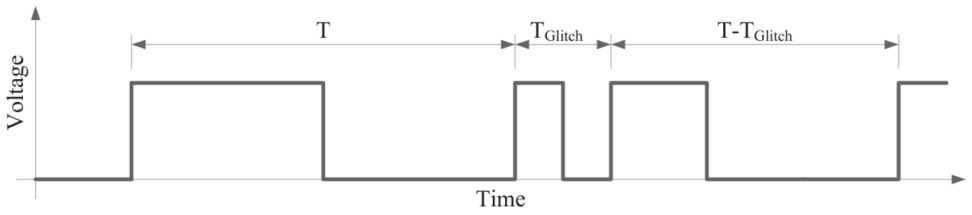

Figure 3.3: Using a FIA to introduce an additional positive clock edge [71].

Korak and Hoefler [71] present clock glitches on the 16-bit AVR ATxmega256 and 32-bit ARM Cortex-M0 microcontroller units (MCUs). While not strictly central mobile processors, MCUs can comprise device subsystems, such as sensor hubs and baseband processors, which were discussed in Chapter 2. The authors were able to inject faults that generated instruction skips with arithmetical (`adds`), branch (`beq` and `breq`), and memory instructions (`ldr` and `str`). It was initially found that pure clock glitches on the Cortex-M0 were ineffective, leading to a combined approach of clock glitches and persistent supply voltage underfeeding. The ARxmega256 did not require voltage underfeeding to generate faults. The attacks affected the *fetch* and *execute* parts of the MCU instruction pipeline, which could be generated with high probability (up to 100%). The test hardware was a Xilinx Spartan-6 XC6SLX45 FPGA with NXP LPC 1114 (ARM Cortex-M0) and ATxmega256A3 extension boards. However, no practical attacks were presented on cryptographic systems or security mechanisms.

Blömer et al. [33] (2014) demonstrate clock FIAs against an implementation of pairing-based cryptography (PBC) from the RELIC toolkit [13] on an Atmel AVR XMEGA A1. Pairings are bilinear maps defined over groups on elliptic curves and underpin several solutions to identity-based encryption (IBE), key agreement, and attribute-based encryption (ABE). The authors discover and exploit a clock glitch vulnerability that triggered the skipping of a jump instruction (`rjmp`) used in the PBC implementation that led to the revealing of the secret key. At most, 4,000 experiments are required to launch the attack (0.025%). It is noted that PBC is not a common cryptographic primitive used in today's mobile security mechanisms, nor is the target device a mobile SoC. However, it is included here for completeness being the only public FIA on PBC.

In 2017, Tang et al. [120] published the CLKScrew attack that uses software-based power management to generate faults that can be leveraged to perform cross-core attacks on TrustZone. The attack functions in much the same way as VoltJockey [98]. CLKScrew exploits DVFS on device SoCs—see §3.2 when examining Qiu et al. [98]—where frequency and voltage regulation manage both the non-secure and secure worlds. The authors use software-based overclocking, which is triggered in the non-secure world, to generate faults in the secure world. This is used to perform two attacks: AES key recovery from a secure world application, and corrupting RSA signature verification used by the secure world OS to verify TAs before they are loaded. CLKScrew has several prerequisites: the attacker must have kernel-mode access to control the power manager, the core clock frequency must be modifiable, interrupts must be disabled, and the TrustZone application can be repeatedly revoked from the non-secure environment to decrypt arbitrary ciphertexts. The work is evaluated on a Google Nexus 6 using a Qualcomm Krait-based SoC. On average, useful faults are generated at a rate of 1.51% for the RSA attack. One in 20 faults (5%) are necessary to introduce a one-byte fault to a desired AES round in order to mount the key recovery attack.

A difficulty with exploiting clock glitches on modern processors is the use of internal phase-locked loop (PLL) circuits (see Figure 3.4). Indeed, we were not able to locate any attacks on the device's external clock in existing literature. The PLL processes the external clock into the system clock frequency, thus acting as a filter. Consequently, traditional (external) clock glitches do not straightforwardly influence the internal clock on today's devices. To counter this, Selmke et al. [113] demonstrated a novel attack against a PLL-equipped ARM Cortex-M0 processor aboard a STMicroelectronics STM32F0308R. The authors successfully perform a clock-based FIA that effectively overclocks the PLL to deliver deterministic perturbations in the PLL's output frequency. Using this, full key recovery is achieved against a software-based AES implementation in ECB mode on the STM32F0308R. In total, 1000 faults were injected of which 16.4% were exploitable.

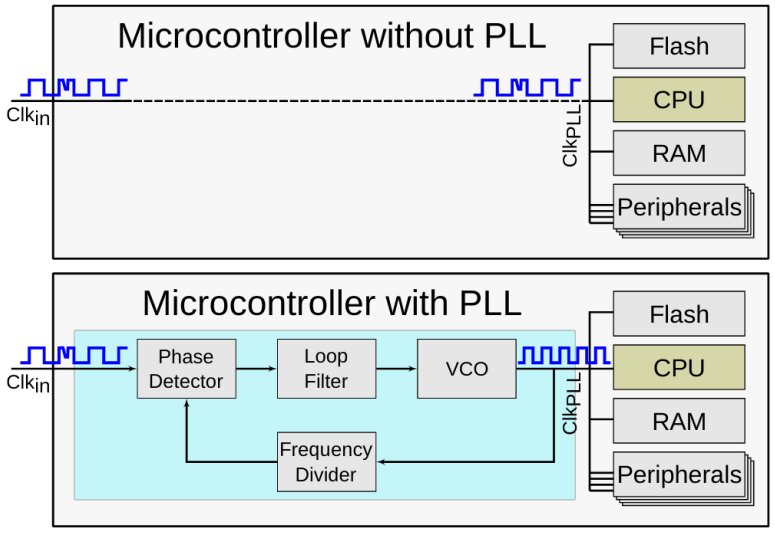

Figure 3.4: A microcontroller with a phase-locked loop (PLL) circuit and one without [113].

3.4 Heating Attacks

A popular FIA vector on embedded devices is expose the device to temperatures beyond its intended operating conditions. Extreme temperatures have been known for over a 15 years to

generate bit errors—often multi-bit errors—in DRAM memory and read and write threshold mismatches in non-volatile memory units [68]. In 2003, Govindavajhala and Appel [59] presented a proof-of-concept showing that temperature-induced bit errors can lead to security vulnerabilites. A 50W light bulb was used to increase the DRAM working temperature of a personal computer to 100°C, which was able to generate up to 10 flipped bits per 32-bit word; successful faults could be used with a 71.4% probability. Using these, the authors develop attacks based on these errors that circumvent the Java type system and introduce security vulnerabilities in two commercial Java virtual machine (JVM) implementations.

Hutter and Schmidt [65] (2013) present heating-based FIAs on an AVR ATmega162 MCU. A low-cost laboratory heating plate is used to heat the MCU to 150°C+, beyond its maximum intended temperature specification of 125°C. The authors discover that the IC begins to exhibit faults between a critical heating window of 152–158°C. RSA-CRT decryptions are then performed at 650ms intervals over a 70 minute period, resulting in 100 faults of which 31 were exploitable using the Boneh et al. [34] RSA-CRT fault attack to recover one of the prime moduli.

In 2014, Korak et al. [70] combined clock glitches with heating attacks in a two-pronged attack on an AVR ATmega162 MCU. They expose the MCU to temperatures of up to 100°C—using the attack setup in Figure 3.5—that facilitates clock glitch attacks presented in earlier work by Balasch et al. [22] on an 8-bit smart card MCU. The authors demonstrate that the clock glitches could be applied with greater success at high temperatures to cause instruction repetition, instruction replacement, and changing destination registers of instructions. However, attacks against particular cryptosystems or security mechanisms were not shown.

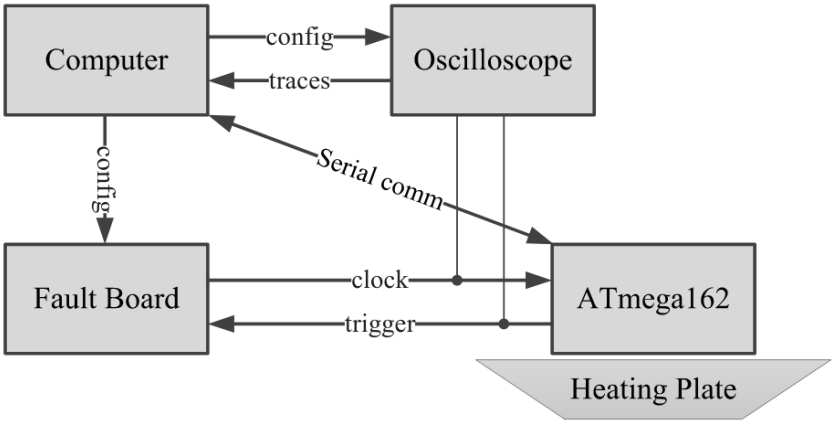

Figure 3.5: Temperature-based attack setup used by Korak et al. [70].

Kumar et al. [72] also present a combined approach like [70], but using extreme temperatures with voltage-based glitches against an ASIC implementation of the Prince block cipher. The authors show that the likelihood of voltage-induced faulty computations increases dramatically when the ASIC is subjected to temperatures beyond 60°C. The outputs of the faults are passed to a software routine that performs the cryptoanalysis and secret key recovery using differential fault analysis. On average, the full attack requires four to five successful faults that occur with a 0.1% probability; thus, the complete attack can be employed using approximately 4,000 to 5,000 fault injection attempts.

We note that no heating attacks have been successfully demonstrated on modern SoCs used by today's mobile devices.

3.5 Electromagnetic FI (EMFI)

Another attack vector is to expose device components to strong electromagnetic pulses. For example, Quisquater and Samyde [103] describe a fault model where Eddy currents induced in the circuit are captured by latches in the target IC, thus generating bit faults. Electromagnetic FIs (EMFIs) have greater spatial precision by targeting particular ICs and shielding other components, and are typically performed through a high precision probe connected to an EM pulse generator (see Figure 3.6). EMFIs on embedded systems have been widely studied since initial attacks in the early 2000s on smart cards [103, 82, 83].

In 2013, Dehbaoui et al. [40] demonstrated an electromagnetic glitch on a software-based AES implementation on a 32-bit ARM Cortex-M3 processor. The authors target the AES round counter to induce the execution of an additional round, which enables feasible cryptanalysis. An EM pulse generator is used to perform the EMFI between the 9th and 10th rounds, triggering effects equivalent to an instruction skip[3] with a high occurrence rate (up to 100%). An attack is presented showing that the encryption key can be recovered with two correct and faulty ciphertext pairs.

Moro et al. [88] (2013) introduce a fault model for 32-bit microprocessors, targeting the ARM Cortex-M3. The authors describe instruction and data corruption vulnerabilities by individually targeting the device's data and instruction buses. More specifically, EMFIs are successfully launched that: (1) modify the values of `LDR` instructions to alter data flow, (2) generate hardware exceptions to alter control flow, (3) replace `STR` instructions, and (4) modify processor registers. Empirical analyses of the effect of EM pulse amplitude on the number and relative frequency of bitset faults on data fetched from flash memory are presented. However, no attacks are demonstrated against cryptographic implementations or security systems.

In 2015, Riviere et al. [105] demonstrated an EMFI that disrupts the instruction cache of an ARM Cortex-M4, which can theoretically be generalised to all ARMv7-M models. The authors demonstrate an EMFI with 96% reproducibility that targets the update of the target processor's prefetch queue buffer. Consequently, using test assembly programs designed for experimentation, four instructions could be skipped and the following four could be replayed. Like [88], however, no attacks are demonstrated against actual cryptographic or security systems. However, it is discussed that the fault model could be applied to CRT-based RSA, AES, and providing privilege escalation on ARMv7-M architectures.

Cui and Housley [39] (2017) present BADFET, which leverages second-order effects of EMFIs. They showed that faults generated in one component can trigger faults in dependent components. A low cost test-bed ($300 USD) is presented for generating EM pulses that corrupt the contents of target DRAM and NAND flash memory, which causes a fault in the CPU's instruction cache. It is here that a large portion of bootloader code is loaded. The authors then demonstrate an attack that faults the secure boot procedure to cause the primary bootloader to skip into an unreachable code region that contains a debugging command line interface (CLI). Using this, the

[3] The authors disclose that a hardware forensic analysis was not conducted of the target processor and, thus, cannot conclude for certain whether an instruction skip actually occurred.

authors load a separate binary that exploits a pre-existing vulnerability in the device's TrustZone SMC implementation (see §2.3.3) to achieve arbitrary code execution in the TEE. A Cisco 8861 IP Phone with a Broadcom BCM11123 SoC is used as the target of evaluation. Following the initial proof-of-concept, the authors were able to replicate the attack 72 out of 100 retries (72%) on the same device.

Liao and Gebotys [77] (2019) describe EMFIs with overclocking for targeting the prefetch stage of an 8-bit MCU (Microchip PIC16F687). The attacks trigger bit-level corruption of opcodes, causing instruction replacement faults within specific programs under evaluation. An exclusive-or (`xorwf`) and move-literal (`movlw`) instruction were particularly vulnerable, which could be replaced with other instructions with a 1.8%–98.7% frequency, including go-to (`goto`), inclusive-or (`iorwf`) and bit-clear (`bcf`). An attack on AES is presented showing that, on average, 222 EM pulses and 5.3 plaintexts were needed for full key recovery. The authors speculate that successful instruction replacements with `goto` could be potentially exploited to bypass authentication checks during a device's secure boot process. However, no experiments were conducted to this end. It is noted that the PIC16F687 required backside decapsulation, which renders the attack invasive.

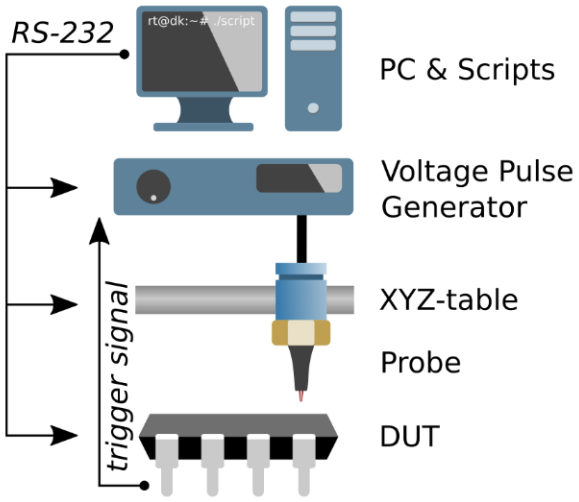

Figure 3.6: EMFI attack setup used by Menu et al. [84].

Menu et al. [84] (2019) present data corruption EMFIs on the transfer bus from flash memory to the 128-bit data buffer of an Atmel SAM3X8E MCU with an ARM Cortex-M3 processor. The attack enables resetting between zero and 128 bits with byte-level precision. It also allows the targeting of the device's pre-fetch mechanism without disturbing or corrupting the code execution. Three components are shown to be vulnerable: the Flash memory, the 128-bit pre-fetch buffer, bus interfaces, and the register file. The authors target a public software implementation of AES and assume the key is stored in Flash memory. In the first attack, an EMFI directly sets or resets the whole key as it is fetched from memory, using the attack setup in Figure 3.6, which was shown to be 100% repeatable. The second study applies an attack by Biham and Shamir [32], which assumes that the attacker can collect ciphertexts of known plaintexts while progressively resetting one bit of the secret key until a zeroed key is reached. The authors leverage this to reduce the key-space of the 128-bit AES key to 16×2^8, which can then be feasibly brute-forced. The last study provides a fault attack that requires a persistent fault in one or more S-box look-up table elements. Any encryption with this corrupted S-box results in a faulty ciphertext if the corrupted entry is accessed during the encryption. The authors show that the AES key entropy can be

reduced to 32-bits when four elements of the S-box are corrupted using EMFIs.

Elmohr et al. [43] (2020) mount EMFIs on an ARM Cortex-M0 on a NXP LPC1114 MCU. An EMFI corruption fault is discovered in the MCU's instruction pipeline. This occurs during the execution stage of a load instruction (`LDR`), thus potentially giving access to unauthorised data. Using a small test program, this fault occurs in cases when a branch (`BNE`) instruction is being fetched at the same time. The faults occur with a frequency of 31%, 13%, 12% and 12% for faulting two, three, four, and five instructions at once respectively. Like [77], the LPC1114 required decapsulation due to its external shielding, thus rendering the FIA invasive and tamper-evident.

Gaine et al. [46] (2020) presented an EMFI privilege escalation attack on a 64-bit SoC using four embedded ARM Cortex-A53 (up to 1.2GHz) CPUs on a mobile development board. The target uses Yocto Project (Sumo)—a Linux distribution tailored to hand-held and embedded devices—with version 4.14 of the Linux kernel. EM pulses were generated using an injection probe upon an XYZ motorised stage, as shown in Figures 3.7 and 3.8. The authors characterised the temporal and spatial requirements of the EMFIs using initial investigations on a test program; only one CPU on the target SoC was sensitive to EMFIs. An instruction skip vulnerability is discovered and exploited on a conditional branch instruction (`cbz`) used in the string comparison (`strcmp`) C function. This is used to bypass the password comparison procedure used by the substitute user Linux command (`su`). After implementing a trigger in `libpam` (the pluggable authentication module), which performs the password procedure, a successful EMFI is performed every two minutes for the same CPU with a fixed frequency and fixed probe position (overall success rate of 2%). The authors are also able to perform the EMFI with DVFS activated. Here, 21 of 6000 FIs were successful (0.35%), equivalent to a success every 300 attempts (15 minutes).

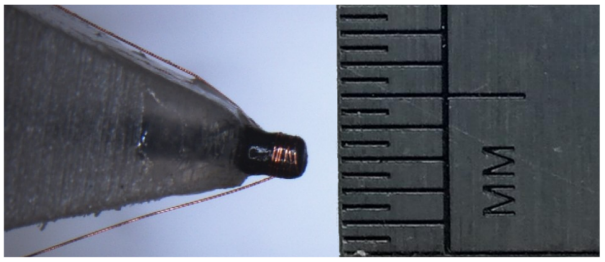

Figure 3.7: EM probe used by Gaine et al. [46].

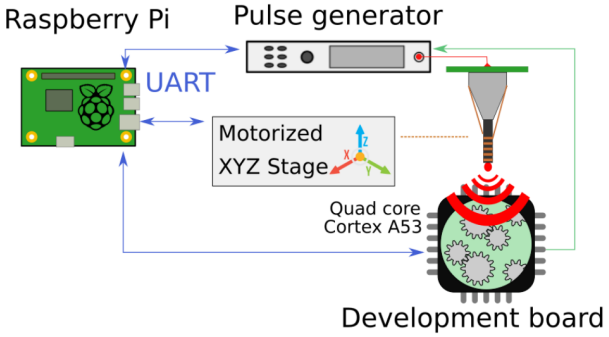

Figure 3.8: High-level attack setup used by Gaine et al. [46].

Chapter 4

Side Channel Attacks (SCAs)

In this chapter, the state of the art of non-invasive side-channel attacks (SCAs) on mobile devices and their related components is surveyed.

4.1 Overview

Rather than identifying and attacking weaknesses of an abstract algorithm, side-channel attacks (SCAs) exploit physical side-effects in its real-world implementation. It is well-known, for example, that vulnerable implementations of widely used cryptographic algorithms can leak information depending on the key and data upon which they are applied. The exploitation of such side-effects have received tremendous attention by the research community for over 20 years, and continue to plague security systems today [37, 56, 130, 131, 134].

Of interest to this report, physical SCAs exploit distinguishable environmental or electrical properties exhibited by devices during execution. Operations can be determined from their power consumption (power analysis), electromagnetic emissions (EM analysis), and more [1, 47, 134]. These properties have been exploited for key recovery from vulnerable cryptographic algorithm implementations, including AES and RSA. Side-channel analysis has also been used for software profiling; that is, discerning the instructions used by firmware or another component, which can assist reverse engineering efforts in black-box settings. While power and EM analysis are the most studied physical side channels, others have been explored in the mobile security literature. These include *smudge attacks*, which exploit residues left on user interfaces to recover user authentication secrets, such as PINs and passwords; *acoustic cryptanalysis*, which leverage differences in sound waves during execution; and *temperature attacks*, which use temperature signatures as a side channel.

In the following sections, the state of the art of physical SCAs is reviewed. The focus is placed on methods that are directly or potentially useful for subverting modern mobile security mechanisms in a non-invasive fashion. This includes power and EM analysis attacks conducted on microcontrollers up to modern system-on-chips, and other relevant side channels that could be applied by a forensic analyst.

4.2 Power Analysis

Power analysis is the study of power variations consumed by the device under investigation to recover information about its internal function. Components contained within today's mobile devices contain millions of transistors that act as voltage switches. During execution, these components are switched on and off, causing measurable voltage fluctuations. Moreover, charge movements through the circuit to other transistor gates, wires etc., produce electromagnetic radiation emissions that can also be measured [69, 60, 93, 86, 85].

Several techniques have been employed for recovering data by statistically analysing power consumption waveforms, or traces. *Simple power analysis* (SPA) directly uses waveforms to identify particular instructions under execution and their corresponding cryptographic operation. These operations can be chained together to determine the data and the secret key upon which the operation is applied. A more advanced technique, first presented by Kocher et al. [69], is *differential power analysis* (DPA). DPA uses statistical analysis of multiple traces and ciphertexts to infer secret data under execution; it is based on computing correlations between the Hamming distance of intermediate cryptographic operations and the power trace to determine key bytes. *Template analysis* involves collecting a set of initial traces and labelling their corresponding operation. Newly measured traces are then classified by mapping it to its closest related template from this set. More recently, machine learning and deep learning has been used to build more complex statistical models for analysing traces and operations. Power analysis side channels have been researched extensively in the smart card and FPGA literature for over 20 years. Yet, despite this, power consumption analysis has found limited utility on mobile SoCs. This section briefly covers recent work on embedded systems for completeness.

One of the main uses of power analysis has been for instruction profiling low-powered devices. This enables an attacker to identify particular instructions under execution on a device. This has a multitude of applications like detecting unauthorised software and reverse engineering black-box programs. In this area, Msgna et al. [89] (2014) showed that power consumption could be used to recognise individual instructions of an ATMega163 microcontroller with 66.78%–100% accuracy using template-based analysis; Figure 4.1 shows a sample one-cycle trace of four ATMega163 instructions. Park et al. [95] (2018) developed a classification model that achieved an instruction recognition rate of 99.03% AVR ATMega328P MCU. However, to the best of our knowledge, similar attacks have not been successfully applied to mobile SoCs.

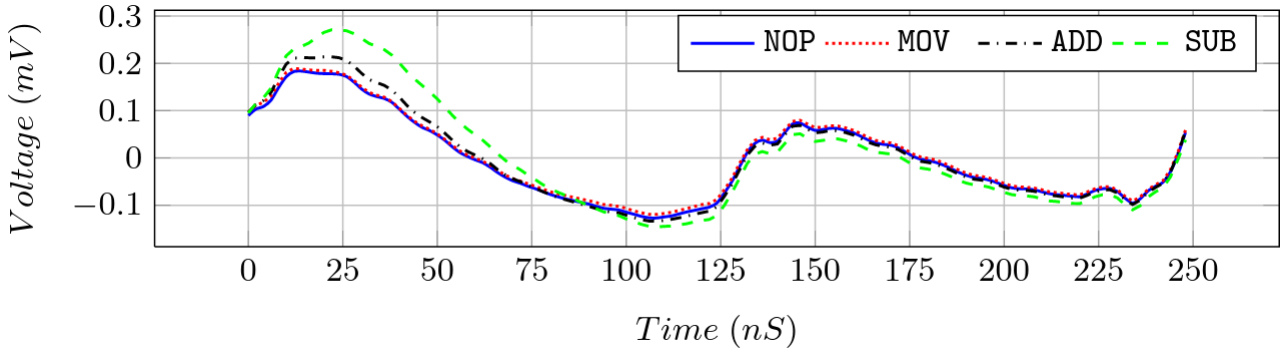

Figure 4.1: Power consumption of the `NOP`, `MOV`, `ADD`, and `SUB` ATMega163 instructions [89].

As stated, machine learning and deep learning has generally superseded traditional power analy-

ses based on carefully crafted statistical models. These techniques can perform feature extraction and classification of thousands, sometimes hundreds of thousands, of traces in a less parametric fashion. Heuser and Zohner [63] (2012) present the use of support vector machines (SVMs), a supervised machine learning algorithm, to perform profiled power analysis. The authors use an 8-bit AVR ATMega-256-1 MCU (8MHz clock) upon which a software-based AES implementation is executed. The AES S-box is used as the profiling target, and the SVM is used to classify the Hamming weight of the S-box value against the power consumption. The authors show that the AES key can be recovered within 20 traces in low-noise environments and up to 60 traces in high-noise environments. Since 2011–2012, a wide variety of work has been published on ML-based SCAs in a number of application areas besides power analysis. The reader is referred to the survey paper by Hettwer et al. [62] for a comprehensive treatment of these works.

Deep learning (DL) methods currently represent the state of the art in power analysis. Deep learning models—neural networks with multiple hidden layers—enable complicated feature extraction from collected traces and the ability to capture complex, non-linear interactions between these features. Maghrebi et al. [80] (2016) presented some of the first results of DL methods applied to AES key recovery. Similar to [63], the authors target an intermediate value of the AES's first-round S-box produced as a function of the plaintext and the secret key. For the first set of experiments, the authors attack an unprotected FPGA-based AES implementation, showing that key recovery can be achieved with 200 traces using a deep convolutional neural network (CNN). Secondly, an unprotected AES implementation aboard a Chipwhisperer is attacked—a development board for side-channel analysis based on an AVR ATMega328P MCU. Using an autoencoder neural network, the first four AES key bytes could be recovered with 20 traces. In the final experiment, the authors develop a masked AES implementation for the Chipwhisperer. In the best case, an autoencoder and CNN was able to recover the secret key with 500 and 1000 traces respectively.

In 2018, Picek et al. [96] explored the use of CNNs and traditional machine learning approaches, such as XGBoost and Random Forests. The authors attack the Hamming weight power consumption of the first AES key byte in the first S-box operation. Protected implementations of AES are also examined, which are collected from a Atmel AVR MCU. The authors use the DPAcontestv2 dataset with 50,000 traces. Using this, a 91.2% test accuracy was achieved when using CNNs to classify the correct Hamming weight. However, it is noted that simpler methods were also used effectively. All evaluated methods—XGBoost, Naïve Bayes, Random Forest, and CNNs—were able to recover the secret key in fewer than 10 traces. A similar approach has been used by Wang [130] (2019), which uses CNNs to perform key recovery against two AVR ATXmega128D4 MCU boards using a 128-bit AES implementation in ECB mode. Notably, the average number of required attack traces for full key recovery differs significantly between the boards during experimentation, requiring 160.3 and 400.2 traces for each of the evaluation boards.

In other power-based SCAs, Schmidt et al. [110] (2010) exploit the small but measurable power consumption of exposed IC input/output pins. This is an alternative method for whenever the direct measurement of power supply lines is not possible, which is often the case on modern mobile devices without invasive intervention. The authors test five devices: an 8-bit Atmel ATMega163 MCU, an 8-bit Atmel AT89S2853 MCU, an NXP LP2148 MCU with a 32-bit ARM ARM7TDMI-S microprocessor, a Virtex-II Pro XC2VP7 FPGA, and an ASIC. Each device contained an implementation of 128-bit AES without any side-channel countermeasures. Plaintexts are sent to the device over a serial interface with an oscilloscope measuring the voltage on the I/O pin. DPAs

are conducted successfully on all five devices; however, the exact number of traces necessary to mount the attacks are not provided.

In 2019, Gnad et al. [56] successfully mounted correlation power analysis attacks against AES-128 on mixed-signal SoCs aboard Internet of Things (IoT) devices. The attack uses power analysis to exploit voltage noise generated by digital subsystems that perturb the signals of on-SoC analog-to-digital converters (ADCs). The threat model is a malicious program aboard the device that wishes to recover the secrets being executed by another. The authors evaluate this side-channel using a ESP32-devkitC, STMicroelectronics STM32L475 IoT Node, and the STMicroelectronics STM32F407VG Discovery. Software is also developed for performing AES-128 for single encryptions of 128-bit plaintext messages, which was implemented using the `mbedTLS` library. Using correlation analysis, ciphertext-based key recovery is successfully performed on the last round of AES using a dataset of 10 million ADC noise traces.

4.3 Electromagnetic Emission Analysis

Using EM analysis to compromise security systems dates back to 1943 when a Bell Telephone engineer discovered oscilloscope perturbations while using an encrypted teletype: the Bell Telephone model 131-B2. The next landmark was the publication of a partially de-classified 1972 NSA paper documenting the beginning of the TEMPEST program[4] [132, 45]. Since then, EM analysis has been researched extensively to break security and cryptographic systems, from smart cards to modern high-frequency system-on-chips. This section looks at the state of the art in EM-based SCAs on mobile devices. Unlike power analysis attacks, EM analysis has been successfully applied in several works to mobile devices.

In 2011, Aboulkassimi et al. [1] presented results of the first EM-based SCA investigation on a modern mobile device. The authors target a software-based AES implementation executing on the Java Platform, Micro Edition (Java ME). Experimentally, trigger signals are sent via the device's microSD card interface, with a commercially available EM probe and oscilloscope used for EM signal acquisition (see Figure 4.2). Two novel techniques are proposed: a spectral density-based approach (SDA) and template-based resynchronisation approach (TRA). These overcome the difficulties in performing EM-based correlation attacks due to the garbage collector and just-in-time (JIT) compiler, which introduce temporal distortions in the EM traces. Using the proposed template-based resynchronisation approach, the authors recovered one byte of the AES key in one-hour after analyzing 250 traces on an undisclosed device with a 32-bit RISC processor and 370MHz clock frequency.

In 2012, Kenworthy and Rohatgi [67] demonstrated side-channel attacks on three undisclosed mobile devices using elliptic curve cryptography (ECC), RSA and AES. The first device—a *"4G LTE smart phone from a major manufacturer"*—is used to perform 2048-bit RSA-CRT encryption, which is a self-written implementation using the square-and-multiply approach. From only a single trace, the authors are able to recover the secret RSA-CRT exponents using a modest test bench—a Yagi antenna, magnetic probe, an ICOM 7000 receiver, and an Ettus Research USRP digitizer—costing $1000 (USD) at the time of writing. The second device, a mobile PDA, is used to perform elliptic curve point multiplication over P-571 using a self-written double-and-sometimes-add implementation. Here, the authors are able to perform full key recovery using a

[4] TEMPEST: Telecommunications Electronics Materials Protected from Emanating Spurious Transmissions.

single trace with an EM probe located approximately three meters (10 feet) away from the device. The third device, *"another mobile phone from a major mobile manufacturer"*, uses the platform AES-128 library to perform AES-CBC encryption on a 200kB data buffer. In this experiment, traces corresponding to 12,500 individual AES block operations were required to perform key recovery using EM-based DPA.

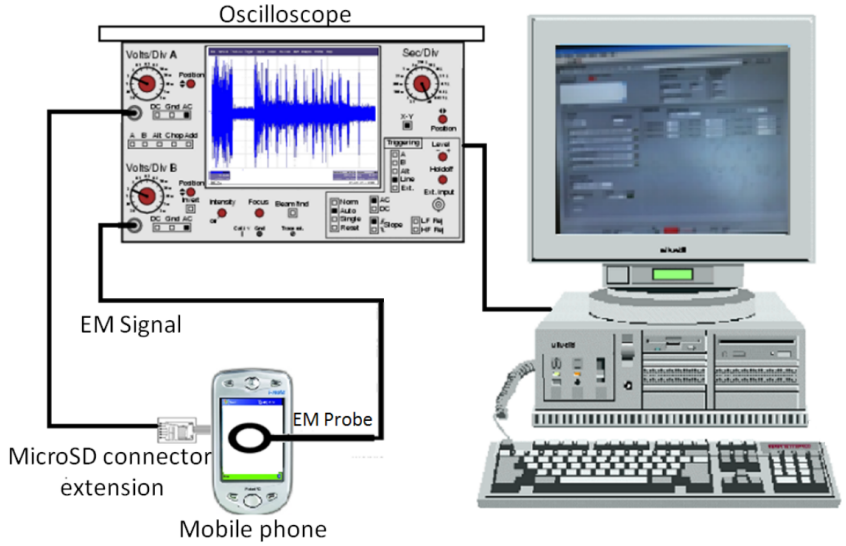

Figure 4.2: Experimental setup used by Aboulkassimi et al. [1].

Montminy et al. [87] (2013) demonstrate the extraction of an AES encryption key running on a 32-bit processor with a 50MHz clock frequency. The device under test is a Stellaris LM4F232H5QD MCU with an ARM Cortex-M4F, which is used to run a software implementation of AES-128 in ECB mode. The main acquisition equipment used is a near-field probe and a software-defined radio to collect EM emission waveforms, including a modified digital television received costing $20 (USD) at the time of publication. Using this, the authors successfully mount a correlation-based analysis attack to extract all AES key bits using 100,000 traces.

Nakano et al. [91] (2014) present SPA attacks on ECC and RSA implementations running on an undisclosed Android smartphone with a clock frequency of 832MHz. The authors target the ECC and RSA implementation provided by the Java Cryptography Extension (JCE) library on Android OS to recover the secret keys. The RSA implementation uses a square-and-multiply approach, which has been shown in much previous work, e.g. [67], to be vulnerable to side-channel analysis. Similarly, ECC double-and-multiply operations are also exploited as in [67]. Analysis at the 10 MHz and 20 MHz frequencies yielded the best results for key recovery, which was achieved in a single trace for both algorithms.

In 2015, Balasch et al. [21] presented EM-based DPA attacks on a 32-bit ARM Cortex-A8 processor running at a 1GHz clock frequency. The authors attack a publicly sourced software-based AES-128 implementation executing on a Texas Instruments AM3358 Sitara SoC on a Beaglebone Black single-board computer (SBC). This system is running a fully-fledged Linux distribution based on Debian 7 with kernel version 3.8.13-bone47. Interestingly, the authors attack both unprotected and masked versions of the algorithm, the latter intended to thwart side-channel analysis. In total, up to 1,200,000 measurements were required to break the masked algorithm using first-order DPA, while 400,000 were required to successfully mount a second-order DPA.

Using the unprotected algorithm, only 10,000 traces were required for full key recovery.

Goller and Sigl [57] (2015) examine the use of radio waves to attack an RSA implementation on five unnamed smartphones[5]. The authors removed the device's shielding plate for stronger EM emanation and used a high-gain antenna attached to a probe that was placed on a capacitor near the main CPU. A C program was developed using the Android SDK that included a square-and-multiply RSA implementation, and the EM waveforms that emanated during its execution were collected using a software-defined radio. The waveforms are shown to clearly correspond to the bits of the key being processed, as shown in Figure 4.4. With the shielding plate installed, 276 traces were required to perform full key recovery with high confidence (0.999 correlation), while 170 traces were required without the shielding plate.

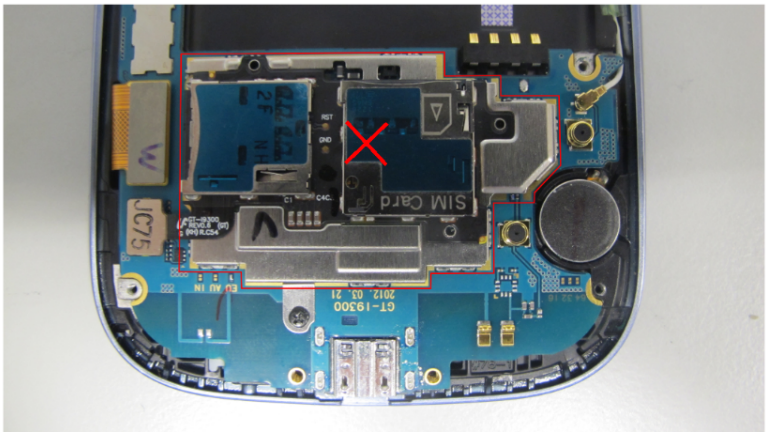

Figure 4.3: Device under test used by Goller and Sigl [57]. The red cross denotes the optimal placement location for the EM probe.

Longo et al. [79] (2015) investigate the feasibility of EM-based DPA against an AES implementation running on a gigahertz CPU. The target platform is a BeagleBone Black SBC with a Texas Instruments AM335x SoC and a 1GHz ARM Cortex-A9 CPU. A software implementation of AES-128 in CBC mode is used as the target cipher, which is taken from OpenSSL; an implementation of AES-256-CBC aboard an undisclosed cryptographic co-processor is also examined. In both cases, DPAs were mounted successfully. For the software implementation, 20,000 traces were required to produce the correct key hypothesis. Meanwhile, the security co-processor required 500,000 traces to be analysed. Notably, of interest to this report, the authors mention that the cryptographic co-processor DPA is likely to be useful against implementations of full-disk encryption (FDE). While no experimental evidence is presented for this use case, it is one of the very few papers to attack a cryptographic co-processor.

In 2016, Belgarric et al. [29] discovered that the elliptic curve digital signature algorithm (ECDSA) in Android's standard cryptographic library was vulnerable to EM analysis. In particular, the authors show that ECC addition and multiplication operations can be distinguished. Full key extraction is demonstrated by placing an electromagnetic probe within an open case of the phone. Traces are triggered via the USB interface and subsequently measured using an oscilloscope. The attack is conducted against an undisclosed phone with a Qualcomm MSM7225 SoC. As a use case, the authors attack and succesfully compromise an Android-based Bitcoin wallet key.

[5]The serial number of one board pictured in the paper corresponds to a Samsung Galaxy S3 (see Figure 4.3).

Experimental evidence is presented showing that up to 39 ECDSA signature traces were required for full key recovery, which took 102 seconds at most.

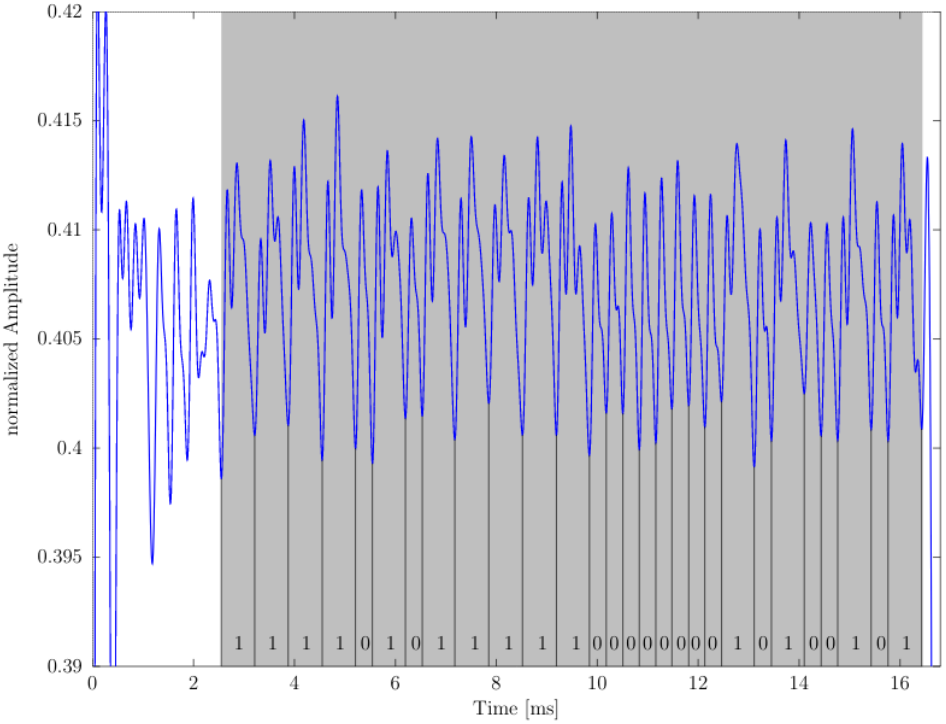

Figure 4.4: Average of 1063 EM traces using the square-and-multiply algorithm. The grey regions show the recovered key bits [57].

Similar results were presented concurrently by Genkin et al. [47] (2016). The authors also demonstrate full key recovery on ECDSA using the OpenSSL implementation on iOS and Android devices. The attack is less invasive than [29], requiring only a magnetic probe placed in proximity of the device, or a power tap on the USB charging cable, and does not require any hardware or software triggers. In the work, ECDSA is exploited experimentally using a Sony-Ericsson Xperia X10 and iPhone 3GS as the devices under test. In total, 5000 signature traces were collected and analysed across the devices, two of which (0.04%) could be exploited to recover the key.

Saab et al. [106] (2016) describe an EM-based DPA attack on a single-board computer with an Intel CPU. In particular, the authors target Intel's AES-NI cryptographic instruction set extensions on the Intel Core i7 Ivy Bridge microprocessor, using an Intel Core i7-3517UE as the target of evaluation. A dedicated application was targeted in C that made calls to Intel's AES-NI sample library (version 1.2), which looped over a group of AES-256 (CBC mode) calls to the assembly routine `iEnc256_CBC`. After collecting 1.5 million traces over 17 days, the authors discover statistically significant Hamming distance leakage, which was due to loading differences in the CPU's cache. A second attack is described that also uses Hamming distance information leakage, but this time caused by the mixing of the round key between successive rounds. For this attack, the authors collected approximately 1.3 million power measurement traces over 22 days.

Bukasa et al. [36] (2017) present the first investigation into EM SCAs on ARM TrustZone. The authors present an empirical analysis of power analysis on an unprotected AES implementation and a PIN verification algorithm. A Raspberry Pi 2 is used as the device under test, which uses

a Broadcom BCM2836 SoC with a quad-core ARM Cortex-A7 at 900MHz. The effect of multi-core versus single-core execution is examined alongside secure-world versus non-secure world execution. A templating approach is employed using 150,000 EM traces as the training set; a success rate of 17.81%–38.30% is then demonstrated depending on the system configuration. The scenario of multi-core execution in the secure world produces the lowest success rate (17.81%), while single-core execution in the non-secure world with the MMU disabled produces the best results (38.30% success).

In 2018, Camurati et al. [37] showed that mixed-signal SoCs—those that contain a CPU and radio transceiver on the same die—can exhibit EM leakages. This occurs because the CPU noise is modulated into the (analog) radio transceiver's emissions, leaving an exploitable and non-invasive attack vector. This technique is evaluated on a Nordic Semiconductor nRF52832, a Bluetooth SoC with a 2.4GHz transceiver and an ARM Cortex-M4 CPU; and a Qualcomm Atheros AR9271, a wireless 802.11N SoC. It is shown that AES keys can be recovered at a distance of 10 meters by observing EM emissions in the 2.4GHz spectrum. The authors use `tinyAES` and `mbedTLS` as two software-based implementations of 128-bit AES. Key recovery is demonstrated on `tinyAES` at a 10 meters distance using 70,000 traces for offline template creation and 428 traces to perform the actual attack using template-based analysis. For `mbedTLS`, a successful attack is mounted at a distance of 1 meter, which required 40,000 traces.

Alam et al. [3] (2018) present the One&Done attack that enables the recovery of RSA encryption keys from a single decryption trace for OpenSSL (version 1.1.0g). Interestingly, the attack is effective even against a 2048-bit fixed-window, constant-time RSA implementation, and when the plaintext message is unknown or randomised (blinded). The attack, which is based on modelling potential control flow transitions of Montgomery multiplications, examines frequencies at 40 MHz around the target devices clock frequency. Two Android mobile phones are used for evaluation: a Samsung Galaxy Centura SCH-S738C with a 800MHz Qualcomm MSM7625A SoC using an ARM Cortex-A5 CPU, and an Alcatel Ideal with a Qualcomm Snapdragon 210 MSM8909 and a quad-core ARM Cortex-A7 CPU. A single-board computer is also evaluate with an Allwinner 13 Cortex-A7 CPU. During experimental validation, 95.7%–99.6% of the target key's bits are successfully recovered depending on the evaluated platform.

In two papers in 2018 and 2020, Benadjila et al. [30, 31] explore deep learning-based analysis of EM traces. The authors present a comprehensive analysis of the application of convolutional neural networks to perform key recovery by classifying the Hamming weights of intermediate cryptographic operations. The target platform is a software implementation of AES executed on an 8-bit AVR microcontroller (ATmega8515) using masked and unmasked versions. A publicly available dataset, the ASCAD dataset, is released to facilitate future research in the domain. Surprisingly, it is found that the common VGG-16 DL network architecture used for image recognition can be effectively applied to side-channel analysis.

Wang et al. [131] (2020) present a deep learning-based SCA on AES from device EM emissions. The authors train three neural networks—two convolutional neural networks (CNNs) with different layer configurations and one multi-layer perceptron (MLP) network—on traces captured from five different Bluetooth devices at five different distances from the target device. Like [37], the Nordic Semiconductor nRF52832 SoC is used along with a 128-bit AES implementation provided by the `tinyAES` C library. 500,000 traces are collected, which are distributed across various distances from the device (wired connection, then 1m–8m) and in different locations (an office and corridor

environment). In the best case, 367 traces of the same encryption were required to recover an AES subkey at a distance of 15m from the device.

4.4 Other Non-Invasive Side Channels

Smudge attacks are another physical side-channel widely researched in the mobile security literature. These attacks, first formalised by Aviv et al. [20] in 2010, exploit residue patterns left by mobile users fingers on mobile touch screens. These residue traces can directly reveal or substantially reduce the search space of mobile-based user authentication secrets, e.g., patterns and PINs (see Figure 4.5). In [20], it is shown that a partial Android pattern was retrievable in approximately 96% of cases without any countermeasures, such as wiping the screen after every attempt. The complete pattern was retrievable in 68% of experimental setups. It is discussed that users tend to select patterns that follow symmetrical variations and other biases, which do not reflect a truly random pattern. In practice, these biases reduce the search space significantly. In the most complex scenarios, where patterns are constructed unnaturally—for example, tight angles and long distances between sequential nodes—smudge residues reduced the possible search space by 50%.

Several countermeasures to smudge attacks have been proposed in the literature, including changing the screen position of the pattern input matrix and colouring the nodes whose position changes between attempts [129]. However, none of these have received widespread commercial deployment, and the Android pattern algorithm is still largely unchanged since the formalisation of the attack class by Aviv et al. [20]. Smudge attacks may offer a relatively primitive, low-cost side channel to access user data aboard touchscreen-equipped mobile devices.

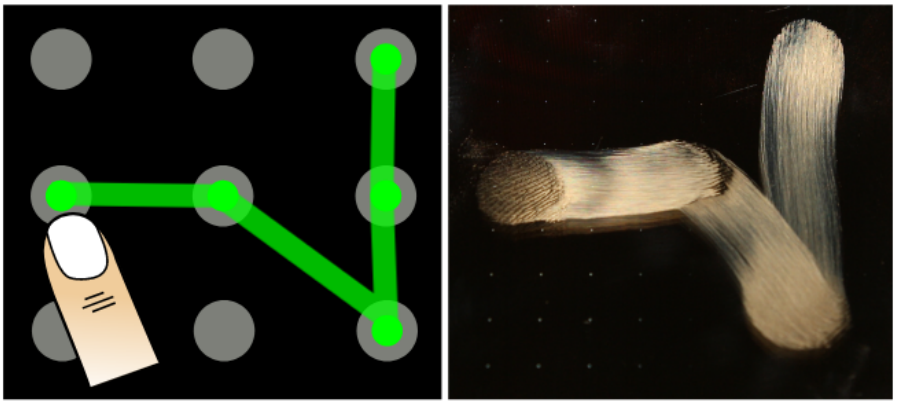

Figure 4.5: Smudge residue left following a pattern-based authentication attempt [129].

Another popular side-channel, albeit with limited application in the mobile security literature, is acoustic cryptanalysis. This involves analysing sound waves produced by a vulnerable security system whose emitted pitch is dependent on the current operation. In 2014, Genkin et al. [48] showed that RSA key extraction using acoustic cryptanalysis is feasible against a Lenovo laptop. The authors target GnuPG's RSA implementation and, using a laboratory microphone setup and a Samsung Note II, show that a 4096-bit key can be recovered within one hour using audible and ultrasonic sound emanations.

Lastly, temperature is another potential side-channel that has also received limited attention in the mobile literature. Hutter and Schmidt [65] (2013) demonstrate that information leakage occurs on an 8-bit AVR ATmega162 MCU by monitoring its temperature using a PT100 sensor element with a 100ms thermal response time. As a proof of concept, the authors use the move (`MOV`) instruction to move all possible values of one input byte (i.e. 256) to 24 internal registers. This is measured for a period of 20 seconds in a loop. The authors demonstrate that the temperature increases depending on the Hamming weight of the processed value. However, it is noted that a full key recovery attack is not shown. Moreover, the attack vector is appropriate only for implementations with long-running operations that create low-frequency signatures. It is unlikely to be directly applicable to the high-frequency processors of today's mobile platforms.

Chapter 5

Evaluation

This chapter contrasts the state-of-the-art of FIAs and SCAs; presents a common comparison framework to compare the requirements, evaluated platforms and attack scenarios for each work; and examines their challenges and limitations.

5.1 State of the Art Comparison

The previous two chapters showed that a wide range of devices and techniques have been deployed in recent literature to attack mobile devices and constituent or closely related components. To compare these works, a set of evaluation criteria is used to compare both attack families using a common framework. The aim of this criteria is to provide the reader with a digestible summary of the requirements, security targets, and attack scenarios that applies to each work. The evaluation criteria used is as follows:

- **Work** and **year**: the work under comparison and its year of publication. Intuitively, older attacks are often evaluated against obsolete platforms that may not generalise to today's modern test devices; for example, countermeasures may have been deployed as a result.
- **Type**: the type of SCA or FIA that was used, such as clock glitch, EMFI, or electromagnetic or power analysis.
- **Evaluation platform**: the platform that was evaluated and was successfully attacked; for example, the device, SoC or microprocessor model. Similarly, it is the case that some works evaluate only a small set of platforms, often only a single one, and may not generalise to modern devices. Moreover, a device of interest may have already been attacked directly in related work.
- **Attack prerequisites**: any preconditions that are necessary for launching the attack, such as having user-space access, accessible hardware, or a particular processor architecture.
- **Implementation**: the implementation targeted by the SCA or FIA. This may be a proprietary OEM implementation, an open-source library, or a bespoke custom implementation developed specifically for the attack. We also evaluate whether the authors integrate a trigger in the implementation to initiate the FIA or the SCA measuring process.
- **Success criteria**: the required number of faults or traces to successfully conduct the attack, or the accuracy of the proposed attack method (if applicable). Attacks that have low success rate might be economically unviable if a substantially large number of attempts are required

to perform a successful SCA or FIA. The examiner may pursue an alternative strategy if the success rate is prohibitively low during preliminary analysis.

- **Attack scenario**: this presents general attack scenarios that are examined in each surveyed work, which are expanded upon in §5.2.

Table 5.1 provides a comprehensive comparative summary using the above criteria for state-of-the-art fault injection attacks surveyed in Chapter 3. A separate comparison is provided in Table 5.2 for side-channel attacks surveyed in Chapter 4.

5.2 General Attack Scenarios

The existing FIA and SCA literature covers a multitude of attack scenarios, such as software-based AES key recovery, privilege escalation, and subverting the secure boot process. To assist practitioners and security researchers, each work in Tables 5.1 and 5.2 has been mapped to one or more of the below attack scenarios depending on their applicability.

AS1 **Secure world privileged access**: gain control of the kernel space of the TEE secure world. This enables privileged access to other valuable targets on the device, including TEE secure world applications in lower protection levels, device drivers to security-critical peripherals, and TEE memory management.

AS2 **Secure world AES/RSA/ECC key recovery**: perform key recovery attacks against security services that use the AES, RSA or ECC cryptosystems in the TEE secure world. Relevant works could be useful when attacking TEE-based key management (keystore) systems and FDE implementations, which were discussed in Chapter 2.

AS3 **Load unauthorised TEE applications**: gain the ability to load unauthorised applications into the TEE. This can be used to further interrogate the TEE secure world to understand its internal operations and attempt privilege escalation attacks against the TEE OS.

AS4 **Bypass secure boot verification checks**: bypass boot-time procedures that enforce the loading of authorised bootloaders, i.e., OEM-signed binaries, to facilitate the loading of unauthorised self-signed or unsigned bootloader images.

AS5 **Non-secure world privileged access**: use a particular attack to gain control of the kernel space of the non-secure world. This enables privileged access to other valuable targets, including device applications in lower protection levels, device drivers, and memory management.

AS6 **AES key recovery**: perform secret key recovery against an encryption/decryption system that uses the Advanced Encryption Standard (AES). As shown in Chapter 2, AES is often used in full-disk encryption and TEE secure world key binding. The reader is referred back to the respective publication in case a particular mode of operation is targeted in the state of the art, e.g., Electronic Code Book (ECB) and Cipher Block Chaining (CBC).

AS7 **RSA key recovery**: perform private key recovery against a security system that uses the RSA algorithm for decryption and digital signature signing. RSA signatures are used widely for data authentication, including bootloader verification during secure boot sequences.

AS8 **ECC key recovery**: perform private key recovery against a security system that uses elliptic curve cryptography (ECC) for decryption and digital signature signing. In particular, the Elliptic Curve Digital Signature Algorithm (ECDSA) is used widely as an alternative to the RSA algorithm.

AS9 **Bypass verification checks**: bypass other run-time security verification steps, such as RSA and ECC signature verification.

AS10 **Load unauthorised data into memory**: corrupt the device's data flow integrity protections to load unauthorised data items, such as a cryptographic keys, during program execution.

AS11 **Key resetting**: perform run-time bit reset attacks on part or all of a cryptographic key to disrupt the security system under investigation.

AS12 **Reverse engineering**: understand the internal operations of black-box software or firmware executing on the device.

AS13 **Mixed-signal system-on-chips (SoCs)**: perform dedicated attacks against devices with mixed-signal SoCs. Such attacks leverage perturbations in analog device signals from co-located security-critical digital components (or vice-versa).

AS14 **Devices with 1Ghz+ clock frequency**: perform a particular attack against devices with high-frequency CPUs (over 1GHz clock frequency), which has been experimentally verified.

AS15 **Devices with multi-core architectures**: perform a particular attack against devices with multi-core CPUs, which has been experimentally verified.

5.3 Challenges and Limitations

In the following sections, challenges and limitations of existing SCA and FIA research is identified and discussed.

5.3.1 Fault Injection Attacks

Low Success Rates

A significant limitation of some voltage- and clock-based FIs is that exploitable faults can be generated at a very low rate. In several works, such faults are generated at a rate of $<$1% of all faults during experimental analysis [122, 123, 33]. This leads to a practicality question: an attack technique could be economically unviable if the cost/time to launch a single fault is significant and a large number of faults are required. A related concern is that some methods require collecting a substantially large set of ciphertexts, which could be impractical depending on the target application. For example, [26] requires up to 2M ciphertexts to perform AES key recovery. Similarly, work in [40] requires AES encryptions to be performed multiple times under the same message and key. This is reasonable if the examiner can replicate these conditions and rapidly inject the fault, otherwise it is likely to suffer the same practical constraints.

Table 5.1: State-of-the-art summary of non-invasive fault injection attacks on mobile and embedded systems.

Work	Year	Type	Evaluation Platform	Prerequisites	Implementation	Success Rate / Criteria	Attack Scenarios
Barenghi et al. [25]	2009	VFI	ARM ARM926EJ-S	—	OpenSSL (v0.9.1i) and Custom	6.6%–39% (RSA-CRT attack) 6.42%–62.77% (e-th Root)	AS7
Barenghi et al. [26]	2010					100K ciphertexts (different plaintexts) 2M ciphertexts (same plaintext)	AS6
Timmers et al. [122]	2016		ARM Cortex-A9 Xilinx Zynq-7010 SoC	AArch32 architecture	Custom ▼	0.01%–0.27%	AS4, AS1
Timmers & Mune [123]	2017		ARM Cortex-A9	AArch32 architecture User-space access	Custom ▼	0.41%–0.63%	AS5
VoltJockey [98]	2019		Qualcomm APQ8084AB SoC Google Nexus 6	SW-controlled PMIC Kernel-space access	Custom ▼	2.2% (AES key recovery) 4.6% (RSA decryption fault)	AS2, AS3
NCC Group [92]	2020		ARM Cortex-A53 MediaTek MT8163V SoC	—	OEM	15.21%–23.44%	AS4
Korak & Hoefler [71]	2014	CFI	AVR ATxmega256 ARM Cortex-M0 NXP LPC 1114		Custom ▼	Up to 100%	AS9, AS6
Blömer et al. [33]	2014		Atmel XMEGA A1		RELIC ▼	0.025%	AS8 (PBC)
CLKScrew [120]	2017		Qualcomm Krait SoC Google Nexus 6	SW-controlled PMIC Kernel-space access Repeated TEE TA invocation	Custom ▼	5% (AES key recovery) 1.51% (RSA verification fault)	AS2, AS3
Selmke et al. [113]	2019		ARM Cortex-M0 STM STM32F0308R	—	Custom ▼	16.4%	AS6 (PLL CPU)
Hutter & Schmidt [65]	2013	HFI + CFI	AVR ATmega162I		Custom	31%	AS7
Korak et al. [70]	2014				Custom ▼	N/A†	AS9, AS6
Dehbaoui et al. [40]	2013	EMFI	ARM Cortex-M3		Custom ▼	Two correct and two faulty plaintext-ciphertext pairs.	AS6
Moro et al. [88]	2013				Custom ▼	N/A†	AS10, AS9
Riviere et al. [105]	2015		ARM Cortex-M4		Custom ▼	Up to 96%	AS6†, AS7†, AS5†
BADFET [39]	2017		Cisco 8861 IP Phone Broadcom BCM11123 SoC	Exploitable TrustZone SMC code	OEM	72%	AS4, AS1
Liao & Gebotys [77]	2019		Microchip PIC16F687	—	Custom ▼	Up to 222 faults required 5.3 plaintexts for AES	AS6, AS9†
Menu et al. [84]	2019		Atmel SAM3X8E ARM Cortex-M3	AES key resides in Flash memory.	OSS from [111] ▼	Up to 100%	AS6, AS11
Elmohr et al. [43]	2020		NXP LPC1114 ARM Cortex-M0 E31 RISC-V SiFive FE310-G002	NXP LPC1114 chip decapsulation	Custom ▼	12%–31%	AS9†, AS10
Gaine et al. [46]	2020		ARM Cortex-A53	—	OSS (`libpam`) ▼	0.35%–2% (DVFS on versus off)	AS5

Gray-coloured works evaluate non-mobile SoCs or devices, e.g. microcontrollers.
*: Device model and manufacturer are not disclosed, †: Discussed as a potential attack, but not experimentally verified.
OEM: OEM implementation, **OSS**: Open-source software, **Custom**: Custom implementation, ▼: Custom trigger implemented.
FI: Fault injection, **VFI**: Voltage-based FI, **CFI**: Clock-based FI, **HFI**: Heating-based FI, **EMFI**: Electromagnetic FI, **SW**: Software, **PMIC**: Power management integrated circuit, **MCU**: Microcontroller unit, **PBC**: Pairing-based cryptography, **PLL**: Phase locked loop, **SoC**: System-on-chip, **DVFS**: Dynamic voltage and frequency scaling.

Table 5.2: State-of-the-art summary of non-invasive physical side-channel attacks on mobile and embedded systems.

Work	Year	Type	Evaluation Platforms	Prerequisites	Implementation	Success Rate / Criteria	Attack Scenarios
Aboulkassimi et al. [1]	2011	EM-SCA	Java ME phone* 32-bit RISC CPU (350MHz)	MicroSD interface	Bouncy Castle and Custom ✣▼	250 traces	AS6
K&R [67]	2012		4G LTE smartphone* Mobile PDA*	Square-and-multiply RSA Double-and-sometimes-add ECC	OEM, OSS and Custom ✣	Single trace (RSA and ECC) 12,500 AES block op. traces	AS7, AS8, AS6
Montminy et al. [87]	2013		Stellaris LM4F232H5QD ARM Cortex-M4F (50MHz)	Accessible hardware components	OEM	100,000 traces	AS6
Nakano et al. [91]	2014		Android smartphone* (832MHz)	Square-and-multiply RSA	OEM	Single trace	AS7, AS8
Balasch et al. [21]	2015		ARM Cortex-A8 (1GHz) TI AM3358 Sitara SoC Beaglebone Black SBC	Accessible hardware components	Custom ▼	400,000 traces (masked AES) 10,000 traces (unmasked AES)	AS6, AS14
Goller & Sigl [57]	2015		Four smartphones* *Samsung Galaxy S3* (see text)	Square-and-multiply RSA External casing is opened	OpenSSL ✣	276 traces (EM shielding) 170 traces (no EM shielding)	AS7, AS6, AS14
Longo et al. [79]	2015		ARM Cortex-A9 (1GHz) TI AM335x SoC Beaglebone Black SBC	Cryptographic co-processor Accessible hardware components	OEM and OpenSSL (v1.0.1j) ▼	20,000 traces (SW AES) 500,000 traces (co-processor AES)	AS6
Belgarric et al. [29]	2016		Android smartphone* Qualcomm MSM7225 SoC	External casing is opened	Bouncy Castle (v1.5) ▼	39 EM traces	AS8
Genkin et al. [47]	2016		Sony-Ericsson Xperia X10 Apple iPhone 3GS	—	OpenSSL (v1.0.x) iOS 7.1.2–8.3	5000 EM traces	AS14, AS8
Saab et al. [106]	2016		Intel i7-3517UE	Intel Ivy Core architecture	Intel AES-NI (Ivy Bridge)	1.3–1.5 million traces	AS6, AS14
Bukasa et al. [36]	2017		Raspberry Pi 2 Broadcom BCM2836 SoC ARM Cortex-A7 (900MHz)	Accessible hardware components	Custom ▼	17.81%–38.30% with 150,000 traces	AS2, AS15
Camurati et al. [37]	2018		Nordic nRF52832 ARM Cortex-M4 Qualcomm Atheros AR9271	Mixed-signal SoC	TinyAES and MbedTLS ✣	70,000 traces (`tinyAES`) 40,000 traces (`mbedTLS`)	AS6, AS13
Alam et al. [3]	2018		Samsung Galaxy Centura Alcatel Ideal	—	OpenSSL (v1.1.0g)	Single trace (95.7%–99.6% recovery)	AS7, AS14
Wang et al. [131]	2020		Nordic nRF52832 SoC		TinyAES ✣▼	367 traces	AS6
Schmidt et al. [110]	2010	P-SCA	NXP LP2148 (ARM ARM7TDMI-S) AVR ATMega163 and AT89S2853		Custom ▼	N/A†	AS6†
H&Z [63]	2012		AVR ATMega-256-1	Accessible power line	Custom ▼	20–60 traces (low–high noise)	AS6
Msgna et al. [89]	2014		AVR ATMega163		Custom	66.78–100% (11 instructions)	AS12
Maghrebi et al. [80]	2016		AVR ATMega328P		Custom	20 traces (unprotected AES) 500 traces (masked AES)	AS6
Picek et al. [96]	2018				Custom	10 traces	AS6
Park et al. [95]	2018				Custom ▼	99.03% (112 instructions)	AS12
Wang [130]	2019		AVR ATXmega128D4		Custom	160.3–400.2 traces	AS6
Gnad et al. [56]	2019		STM32L475 and STM32F407VG ESP32-devkitC	Mixed-signal SoC	TinyAES and MbedTLS✣▼	10 million traces	AS6, AS13
Genkin et al. [48]	2014	AC-SCA	Samsung Note II Lenovo laptop	—	GnuPG (v1.x)	One-hour audio trace	AS7

Gray-coloured works evaluate non-mobile SoCs or devices, e.g. microcontrollers.
*: Device model and manufacturer are not disclosed, †: Discussed as a potential attack, but not experimentally verified.
OEM: Proprietary OEM implementation, **OSS**: Open-source software implementation, **Custom**: Custom implementation, ▼: Software trigger used, ✣: Implementation and/or version not disclosed.
SCA: Side-channel attack, **EM-SCA**: Electromagnetic SCA, **P-SCA**: Power-analysis SCA, **AC-SCA**: Acoustic SCA, **JVM**: Java virtual machine, **Java ME**: Java Platform – Micro Edition, **JCE**: Java cryptography extensions library, **ECC**: Elliptic curve cryptography, **SBC**: Single-board computer, **SoC**: System-on-chip.

Risk of Device Damage

An overarching issue with FIAs is the risk of irreparably destroying the device under test, thus losing any data of interest to the examiner. By definition, such attacks leverage unexpected behaviour when the device is subjected to conditions beyond the manufacturer's guidelines. This is likely to occur during the search process of identifying effective glitch parameters for fault characterisation, shown previously in Figure 3.3 in Chapter 3 for clock glitch attacks. In particular, extreme over- or under-volting, also known as 'over-glitching', can inflict permanent and unpredictable damage on device components, sometimes in ways that are not immediately observable [28].

FIA Countermeasures

A range of hardware- and software-based countermeasures to FI attacks have been developed and deployed in recent years. In hardware, DC power filters have been known to provide effective defenses against voltage-based FIs for over 20 years since their deployment on commercial smart cards [68]. While not strictly a security countermeasure, phase-locked loop (PLL) circuits have also increased the difficulty of launching clock glitch attacks, which processes the external clock into the internal CPU clock frequency [113]. Dedicated frequency detection circuits have also been proposed for hardware clock glitches [24], while hardware frequency locking can be deployed to thwart software-based clock glitch attacks like CLKScrew [120]. In the case of EMFIs, tamper-resistant EM shielding can be used to minimise EM emissions during execution, while on-board device sensors have been shown to be an effective method for detecting heating-based FIAs [10].

Several software countermeasures are also known widely in the literature. These include double or multiple checks of security-critical procedures, such as signature verification, that force the attacker to perform several FIAs to bypass a particular check. Other redundancies have been proposed in software and in hardware for computing the same result multiple times and checking their equivalence, as well as execution and timing randomisation [24]. As stated previously, published research papers do not typically consider a wide range of hardware and software countermeasures, which may limit their applicability. An observation is that these countermeasures generally increase the difficulty of performing FIAs and do not thwart them definitively.

5.3.2 Side-Channel Attacks

Inaccessible Hardware

Hardware access is a major challenge to conducting certain classes of SCAs on mobile devices, especially power-based SCAs. Voltage supply pins are not easily accessible on modern SoCs because the pins are occluded by the package itself. Invasive techniques are thus required just to investigate the feasibility of an attack, let alone execute one successfully. Moreover, conducting power analysis on BGA packages requires specialist equipment and expertise in itself, which increases the attack complexity significantly [21]. It is likely for this reason that security researchers have tended to focus on power-based SCAs on platforms with easily accessible supply pins, such as single-board computers (SBCs) and microcontroller units.

SCA Countermeasures

Similar to FIAs, software and hardware countermeasures to side-channel attacks have been well-studied in the literature. Introducing randomness into the program execution and the clock frequency are both common defense hardware strategies. Constant-time operations, such as the ECC Montgomery ladder, are the main software countermeasure, which do not leak secret-dependent information. Masked implementations of AES device are another countermeasure to defend against differential power analysis attacks; however, work already in this report shows that these can still be attacked, albeit with greater complexity. EM shielding can provide an effective defense against exploitable EM emissions, such as Faraday cage packages [104]. Although these countermeasures are widely known, researchers have still tended to focus on unprotected hardware and software in recent years. For example, [67] attacks double-and-sometimes-add ECC, which is known to be side-channel vulnerable. Consequently, it is likely that difficulties will be faced by security researchers in replicating such work against practical systems and well-known cryptographic libraries that contain side-channel countermeasures.

5.3.3 Discussion

During the research of this report, one immediate limitation of FIAs and SCAs is the extent to which published attacks and methods can be generalised to other evaluation platforms. This includes those that have not yet reached the marketplace. Published research is typically constrained to a small set of evaluation platforms; often only a single target is used, as is the case with $>$80% state-of-the-art FIA and $>$60% SCA works. Differences in processor architectures, power management systems/voltage regulators, and external clocks may significantly affect attack feasibility to the point of being ineffective. As such, it is potentially the case that the success rates/criteria of published techniques—as shown in Tables 5.1 and 5.2—may represent a best-case scenario if alternative platforms are under test.

Besides the tendency for researchers to evaluate a small set of test devices, another limitation is the heavy use of custom device triggers integrated in the target implementation. The vast majority of FIAs, as shown in Table 5.1, require a custom trigger from the device to perform the fault injection, as do a large number of SCA works. Such triggers are unlikely to exist, or can be easily integrated, into implementations on commercial devices. Related to this is the reluctance for authors to publicly publish their custom implementations. This is further compounded by the lack of publicly available data sheets regarding modern mobile devices and their components, e.g., SoCs, As such, a forensic analyst may encounter great difficulty in replicating attacks on off-the-shelf devices.

Furthermore, methodological issues have also been identified as a source of reproducibility issues. Benadjila et al. [30] state that, for the most advanced SCAs, the data analysis framework is often not published: *“hyperparameterisation [of machine learning models] has often been kept secret by the authors who only discussed the main design principles and on the attack efficiencies”*. The disclosure of model hyperparameters is a key step towards reproducing published results. Furthermore, Wang et al. [131] showed that the effectiveness of machine and deep learning based SCAs can vary severely between devices. They show experimentally that a model that is trained on power traces for performing key recovery from one board does not generalise to others, even those with the same ICs. More specifically, a DL model with a purported 96% accuracy drops to only 2.45% when tested on traces captured from another board with the same IC. Lastly,

it is observed that many authors simply do not disclose the devices under test [1, 57, 67, 91]. In many cases, only high-level details are shared, e.g., *"an Android smartphone"* [91] or a *"4G LTE smart phone from a major manufacturer"* [67]. Obscuring precise platform information can significantly hinder reproducibility significantly if the attack approach does not generalise well.

The prevalence of ever-more advanced processor architectures also continues to be a major barrier to SCAs and FIAs. Multi-core, high-frequency CPUs with complex instruction pipelines and memory management have increased attack complexity significantly over the past 10 years [1, 47, 91]. For FIAs, it necessitates the injection of extremely precise faults during program execution, which is limited by the precision of the testing equipment being used. Similarly, for SCAs, the researcher must account for context switching, garbage collectors, virtualisation, multi-stage pipelines, and other features that can disrupt the collection of exploitable power and EM traces. In particular, we are not aware of any attacks on complex PoP SoCs where multiple packages are integrated into a single unit. The growing utilisation of TEEs complicates this further, which share the same physical hardware as the native OS in a time-sliced fashion. Consequently, today's security researcher faces several barriers when accounting for parallel execution if only a single application is of interest.

Finally, we observe that several previously effective methods on constrained devices, such as microcontrollers, have found limited utility on mobile devices. Heating fault injections, power analysis, EMFIs, and clock glitches in particular have not been widely applicable to mobile devices. The wide deployment of PLL circuits that separate the device's internal clock from its external clock, the difficulty in accessing supply voltage pins to measure power consumption, and complex high-frequency SoCs that make it difficult to introduce faults successfully have contributed to this trend [44, 91, 113].

5.4 Future Directions

Deep learning-based approaches to side-channel analysis have rose to prominence in the last 12-24 months. These methods enable complex feature extraction from underlying data, such as power traces, and the ability to model highly non-linear interactions for classification. This is in contrast to traditional statistical methods that use carefully chosen parametric models. At present, DL-based SCAs are attracting significant attention by researchers, with promising results being presented in leading security conferences and journals. It is likely that this trend will persist given the new avenues it opens for EM- and power-based SCAs without defining highly bespoke statistical models.

The growth of mixed-signal and multi-SoC architectures are an important development in the development of fault injection and side-channel attacks on mobile devices. Recent research, e.g. [37] and [56], has shown that these can offer new attack vectors into recovering secret data. Given the economic incentives to shrink device hardware into ever-more compact units, it is likely that this trend will continue, and that new attack avenues may be opened from the interaction between increasingly diverse on-SoC components. In parallel, manufacturers continue to delegate long-running but low-complexity device features, such as sensor hubs, to separate device SoCs. This allows the primary SoC with the device's application processor to remain in a low-powered state while low-complexity processing is handled by a less energy-intensive SoC. Similarly, the

interaction between subsystem SoCs could offer new attack vectors in future research.

Security practitioners should also be aware of emerging methods and applications of using mobile TEEs, a multitude of which have been published in recent research. Examples include secure mobile-based deep learning [73], authenticating mobile adverts from advertising networks [76], protecting system log integrity [115, 66], new paradigms of remote TEE-to-TEE communication [116, 117], securing health-related data [112], confidential image processing [35], and executing cryptocurrency wallet operations [49]. These proposals could influence commercial services that store and process high-value assets, and may serve as future attack targets.

Chapter 6

Conclusion

The difficulty of mobile data extraction and analysis methods has increased significantly due to the growing complexity of today's mobile platforms and the increased focus on security. From a hardware perspective, the development of system-on-chips with multi-core CPUs, gigahertz clock speeds, and hardware security extensions contribute to this trend. From a software viewpoint, the widespread deployment of full-disk encryption, secure boot chains, and TEEs to protect potentially valuable evidence are all major barriers to forensic investigations.

This report has presented an extensive survey of the state of the art in physical fault injection and side-channel attacks, or FIAs and SCAs respectively. These techniques can be conducted without full knowledge of the information that is used to protect device data on today's devices, such as user authentication codes to deactivate full-disk encryption and decrypt TEE-resident data. Over 40 research publications were examined, published between 2009 and 2020, from which 15 attack scenarios were drawn. Each research publication has been individually mapped to the relevant attack scenario, alongside any prerequisites, the evaluated platforms, and their published success rates.

It is hoped that his report provides an informative and comprehensive summary of the applicability of state-of-the-art attacks on today's mobile devices. Moreover, this report presented some challenges and limitations pertaining to FIAs and SCAs. In particular, it has been shown that the reproducibility of attacks remains an issue, while low success rates and a small number of evaluated platforms could lead to generalisation issues on new and alternative devices. Furthermore, it was discussed that much research does not take into consideration a large suite of FIA or SCA countermeasures that may be used on today's mobile devices. Lastly, future directions were provided to inform the reader of potential developments in the state of the art.

Chapter 7

List of Abbreviations

Abbreviation	Translation
AES	Advanced Encryption Standard
ADC	Analog-to-Digital Converter
AP	Application Processor
API	Application Programming Interface
ART	Android Runtime
ASIC	Application Specific Integrated Circuit
BGA	Ball Grid Array
BP	Baseband Processor
CBC	Cipher Block Chaining
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CRT	Chinese Remainder Theorem
DES	Data Encryption Standard
DL	Deep Learning
DPA	Differential Power Analysis
DSP	Digital Signal Processor
DVFS	Dynamic Voltage and Frequency Scaling
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellmann
ECDSA	Elliptic Curve Digital Signature Algorithm
EM	Electromagnetic
EMFI	Electromagnetic Fault Injection
eMMC	Embedded Multi-Media Card
FDE	Full-Disk Encryption
FIA	Fault Injection Attack
FPGA	Field-Programmable Gate Array
GPIO	General-Purpose Input/Output
GPU	Graphics Processing Unit
HMAC	Hash-based Message Authentication Code
HUK	Hardware Unique Key
I/O	Input/Output

Abbreviation	Translation
I2C	Inter-Integrated Circuit
IC	Integrated Circuit
IP	Intellectual Property
JTAG	Joint Test Action Group
KDF	Key Derivation Function
LEA	Law Enforcement Agency
ML	Machine Learning
MMU	Memory Management Unit
NFC	Near-Field Communication
OEM	Original Equipment Manufacturer
OS	Operating System
PCB	Printed Circuit Board
PLL	Phase-Locked Loop
PoP	Package-on-Package
POST	Power-On Self-Test
RAM	Random Access Memory
REE	Rich Execution Environment
RFID	Radio-Frequency Identification
ROM	Read-Only Memory
RoT	Root of Trust
RSA	Rivest–Shamir–Adleman
RTOS	Real-Time Operating System
SCA	Side-Channel Attack
SEP	Secure Enclave Processor
SGX	Software Guard Extensions
SoC	System-On-Chip
SPA	Simple Power Analysis
SPI	Serial Peripheral Interface
TA	Trusted Application
TEE	Trusted Execution Environment
TOE	Target Of Evaluation
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
XEX	Xor-Encrypt-Xor
XTS	XEX-based Tweaked-Codebook Mode With Ciphertext Stealing

Bibliography

[1] Driss Aboulkassimi, Michel Agoyan, Laurent Freund, Jacques Fournier, Bruno Robisson, and Assia Tria. Electromagnetic analysis (EMA) of software AES on Java mobile phones. In *2011 IEEE International Workshop on Information Forensics and Security*, pages 1–6. IEEE, 2011.

[2] Michel Agoyan, Jean-Max Dutertre, David Naccache, Bruno Robisson, and Assia Tria. When clocks fail: On critical paths and clock faults. In *International Conference on Smart Card Research and Advanced Applications*, pages 182–193. Springer, 2010.

[3] Monjur Alam, Haider Adnan Khan, Moumita Dey, Nishith Sinha, Robert Callan, Alenka Zajic, and Milos Prvulovic. One&Done: A single-decryption EM-based attack on OpenSSL's constant-time blinded RSA. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 585–602, 2018.

[4] Anandtech. Samsung Announces Exynos 980, 2019. https://www.anandtech.com/show/14829/samsung-announces-exynos-980-midrange-with-integrated-5g-modem.

[5] Ross Anderson and Markus Kuhn. Low cost attacks on tamper resistant devices. In *International Workshop on Security Protocols*, pages 125–136. Springer, 1997.

[6] Android. Full-disk encryption, 2020. https://source.android.com/security/encryption/full-disk.

[7] Android. Gatekeeper, 2020. https://source.android.com/security/authentication/gatekeeper.

[8] Android. Keystore System, 2020. https://developer.android.com/training/articles/keystore.

[9] Android. Platform Architecture, 2020. https://developer.android.com/guide/platform.

[10] Md Toufiq Hasan Anik, Jean-Luc Danger, Sylvain Guilley, and Naghmeh Karimi. Detecting failures and attacks via digital sensors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[11] Apple, Inc. iOS Security: iOS 11, 2018. https://www.apple.com/ca/business-docs/iOS_Security_Guide.pdf.

[12] Apple, Inc. Apple Platform Security, 2020. https://manuals.info.apple.com/MANUALS/1000/MA1902/en_US/apple-platform-security-guide.pdf.

[13] D. F. Aranha and C. P. L. Gouvêa. RELIC cryptography toolkit, 2014. https://code.google.com/archive/p/relic-toolkit/.

[14] G. Arfaoui, S. Gharout, and J. Traoré. Trusted execution environments: A look under the hood. In *2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 259–266, 2014.

[15] ARM Holdings. Introducing 2017's extensions to the ARM architecture, 2017. https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/introducing-2017s-extensions-to-the-arm-architecture.

[16] ARM Holdings. ARMv8-A Trusted Board Boot Requirements, 2018. https://static.docs.arm.com/den0006/d/DEN0006D_Trusted_Board_Boot_Requirements.pdf.

[17] ARM Holdings. Arm Mali Graphics Processing Units (GPUs), 2020. https://developer.arm.com/ip-products/graphics-and-multimedia/mali-gpus.

[18] ARM Holdings. Processor IP for the Widest Range of Devices, 2020. https://www.arm.com/products/silicon-ip-cpu.

[19] ARM Holdings. Security IP: Layered Security for Your Next SoC, 2020. https://www.arm.com/products/silicon-ip-security.

[20] Adam J Aviv, Katherine L Gibson, Evan Mossop, Matt Blaze, and Jonathan M Smith. Smudge attacks on smartphone touch screens. *Workshop on Offensive Technologies*, 10:1–7, 2010.

[21] Josep Balasch, Benedikt Gierlichs, Oscar Reparaz, and Ingrid Verbauwhede. DPA, bitslicing and masking at 1 GHz. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 599–619. Springer, 2015.

[22] Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. An in-depth and black-box characterization of the effects of clock glitches on 8-bit MCUs. In *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 105–114. IEEE, 2011.

[23] Feng Bao, Robert H Deng, Yongfei Han, A Jeng, A Desai Narasimhalu, and T Ngair. Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults. In *International Workshop on Security Protocols*, pages 115–124. Springer, 1997.

[24] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, 2006.

[25] Alessandro Barenghi, Guido Bertoni, Emanuele Parrinello, and Gerardo Pelosi. Low voltage fault attacks on the RSA cryptosystem. In *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 23–31. IEEE, 2009.

[26] Alessandro Barenghi, Guido M Bertoni, Luca Breveglieri, Mauro Pellicioli, and Gerardo Pelosi. Low voltage fault attacks to AES. In *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 7–12. IEEE, 2010.

[27] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proceedings of the IEEE*, 100(11):3056–3076, 2012.

[28] Alessandro Barenghi, Cédric Hocquet, David Bol, François-Xavier Standaert, Francesco Regazzoni, and Israel Koren. Exploring the feasibility of low cost fault injection attacks on sub-threshold devices through an example of a 65nm AES implementation. In *International Workshop on Radio Frequency Identification: Security and Privacy Issues*, pages 48–60. Springer, 2011.

[29] Pierre Belgarric, Pierre-Alain Fouque, Gilles Macario-Rat, and Mehdi Tibouchi. Side-channel analysis of Weierstrass and Koblitz curve ECDSA on Android smartphones. In *Cryptographers' Track at the RSA Conference*, pages 236–252. Springer, 2016.

[30] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. *ANSSI, France & CEA, LETI, MINATEC Campus, France*, 22:2018, 2018.

[31] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *Journal of Cryptographic Engineering*, 10(2):163–188, 2020.

[32] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Annual international cryptology conference*, pages 513–525. Springer, 1997.

[33] Johannes Blömer, Ricardo Gomes Da Silva, Peter Günther, Juliane Krämer, and Jean-Pierre Seifert. A practical second-order fault attack against a real-world pairing implementation. In *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 123–136. IEEE, 2014.

[34] Dan Boneh, Richard A DeMillo, and Richard J Lipton. On the importance of checking cryptographic protocols for faults. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 37–51. Springer, 1997.

[35] Tiago Brito, Nuno O Duarte, and Nuno Santos. ARM TrustZone for secure image processing on the cloud. In *2016 IEEE 35th Symposium on Reliable Distributed Systems Workshops (SRDSW)*, pages 37–42. IEEE, 2016.

[36] Sebanjila Kevin Bukasa, Ronan Lashermes, Hélène Le Bouder, Jean-Louis Lanet, and Axel Legay. How TrustZone could be bypassed: Side-channel attacks on a modern system-on-chip. In *IFIP International Conference on Information Security Theory and Practice*, pages 93–109. Springer, 2017.

[37] Giovanni Camurati, Sebastian Poeplau, Marius Muench, Tom Hayes, and Aurélien Francillon. Screaming channels: When electromagnetic side channels meet radio transceivers. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 163–177, 2018.

[38] Zitai Chen, Georgios Vasilakis, Kit Murdock, Edward Dean, David Oswald, and Flavio D. Garcia. VoltPillager: Hardware-based fault injection attacks against intel SGX enclaves using the SVID voltage scaling interface. In *30th USENIX Security Symposium*. USENIX Association, 2021.

[39] Ang Cui and Rick Housley. BADFET: Defeating modern secure boot using second-order pulsed electromagnetic fault injection. In *11th USENIX Workshop on Offensive Technologies*, 2017.

[40] Amine Dehbaoui, Amir-Pasha Mirbaha, Nicolas Moro, Jean-Max Dutertre, and Assia Tria. Electromagnetic glitch on the AES round counter. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 17–31. Springer, 2013.

[41] Munkhzorig Dorjmyagmar, MinChang Kim, and Hyoungshick Kim. Security analysis of Samsung KNOX. In *19th International Conference on Advanced Communication Technology (ICACT)*, pages 550–553. IEEE, 2017.

[42] Jan-Erik Ekberg, Kari Kostiainen, and N Asokan. The untapped potential of trusted execution environments on mobile devices. *IEEE Security & Privacy*, 12(4):29–37, 2014.

[43] M. A. Elmohr, H. Liao, and C. H. Gebotys. EM fault injection on ARM and RISC-V. In *21st International Symposium on Quality Electronic Design (ISQED)*, pages 206–212, 2020.

[44] European Commission. *D1.1. – State of the Art in Mobile Forensics*. The European Commission, 2020.

[45] Jeffrey Friedman. Tempest: A signal problem. *NSA Cryptologic Spectrum*, 35:76, 1972.

[46] Clément Gaine, Driss Aboulkassimi, Simon Pontié, Jean-Pierre Nikolovski, and Jean-Max Dutertre. Electromagnetic fault injection as a new forensic approach for SoCs. In *2020 IEEE International Workshop on Information Forensics and Security*, pages 1–6. IEEE, 2020.

[47] Daniel Genkin, Lev Pachmanov, Itamar Pipman, Eran Tromer, and Yuval Yarom. ECDSA key extraction from mobile devices via nonintrusive physical side channels. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1626–1638, 2016. https://www.cs.tau.ac.il/~tromer/mobilesc/mobilesc.pdf.

[48] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In *Annual Cryptology Conference*, pages 444–461. Springer, 2014.

[49] Miraje Gentilal, Paulo Martins, and Leonel Sousa. TrustZone-backed bitcoin wallet. In *Proceedings of the Fourth Workshop on Cryptography and Security in Computing Systems*, pages 25–28, 2017.

[50] GlobalPlatform. TEE Client API (v1.0), 2010.

[51] GlobalPlatform. TEE System Architecture (v1.2), 2017.

[52] GlobalPlatform. TEE Internal Core API (v1.2.1), 2019.

[53] GlobalPlatform. Technology Document Library, 2020. http://globalplatform.org/specs-library/?filter-committee=tee.

[54] GlobalPlatform. TEE Management Framework (TMF) (v1.1.2), 2020.

[55] GlobalPlatform. TEE Protection Profile (v1.3), 2020.

[56] Dennis Gnad, Jonas Krautter, and Mehdi B Tahoori. Leaky noise: new side-channel attack vectors in mixed-signal IoT devices. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 305–339, 2019.

[57] Gabriel Goller and Georg Sigl. Side channel attacks on smartphones and embedded devices using standard radio equipment. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 255–270. Springer, 2015.

[58] Google. Titan M makes Pixel 3 our most secure phone yet, 2020. https://blog.google/products/pixel/titan-m-makes-pixel-3-our-most-secure-phone-yet/.

[59] Sudhakar Govindavajhala and Andrew W Appel. Using memory errors to attack a virtual machine. In *Symposium on Security and Privacy*, pages 154–165. IEEE, 2003.

[60] Yi Han, Ioannis Christoudis, Konstantinos I Diamantaras, Saman Zonouz, and Athina Petropulu. Side-channel-based code-execution monitoring systems: a survey. *IEEE Signal Processing Magazine*, 36(2):22–35, 2019.

[61] Ludger Hemme. A differential fault attack against early rounds of (triple-)DES. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 254–267. Springer, 2004.

[62] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Applications of machine learning techniques in side-channel attacks: A survey. *Journal of Cryptographic Engineering*, pages 1–28, 2019.

[63] Annelie Heuser and Michael Zohner. Intelligent machine homicide: Breaking cryptographic devices using support vector machines. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 249–264. Springer, 2012.

[64] Huawei. iTrustee, 2020. https://www.huawei.com/en/sustainability/stable-secure-network/privacy-protection.

[65] Michael Hutter and Jörn-Marc Schmidt. The temperature side channel and heating fault attacks. In *International Conference on Smart Card Research and Advanced Applications*, pages 219–235. Springer, 2013.

[66] Vishal Karande, Erick Bauman, Zhiqiang Lin, and Latifur Khan. SGX-Log: Securing system logs with SGX. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 19–30, 2017.

[67] Gary Kenworthy and Pankaj Rohatgi. Mobile device security: The case for side channel resistance. *Mobile Security Technologies*, 2012.

[68] C. H. Kim and J. Quisquater. Faults, injection methods, and fault attacks. *IEEE Design Test of Computers*, 24(6):544–545, 2007.

[69] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual international cryptology conference*, pages 388–397. Springer, 1999.

[70] T. Korak, M. Hutter, B. Ege, and L. Batina. Clock glitch attacks in the presence of heating. In *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 104–114, 2014.

[71] Thomas Korak and Michael Hoefler. On the effects of clock and power supply tampering on two microcontroller platforms. In *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 8–17. IEEE, 2014.

[72] Raghavan Kumar, Philipp Jovanovic, and Ilia Polian. Precise fault-injections using voltage and temperature manipulation for differential cryptanalysis. In *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, pages 43–48. IEEE, 2014.

[73] Roland Kunkel, Do Le Quoc, Franz Gregor, Sergei Arnautov, Pramod Bhatotia, and Christof Fetzer. TensorSCONE: A secure TensorFlow framework using Intel SGX. *arXiv preprint arXiv:1902.04413*, 2019.

[74] Paul Leignac, Olivier Potin, Jean-Baptiste Rigaud, Jean-Max Dutertre, and Simon Pontié. Comparison of side-channel leakage on rich and trusted execution environments. In *Proceedings of the 6th Workshop on Cryptography and Security in Computing Systems*, pages 19–22, 2019.

[75] Huiyun Li, Guanghua Du, Cuiping Shao, Liang Dai, Guoqing Xu, and Jinlong Guo. Heavy-ion microbeam fault injection into SRAM-based FPGA implementations of cryptographic circuits. *IEEE Transactions on Nuclear Science*, 62(3):1341–1348, 2015.

[76] Wenhao Li, Haibo Li, Haibo Chen, and Yubin Xia. Adattester: Secure online mobile advertisement attestation using TrustZone. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 75–88, 2015.

[77] Haohao Liao and Catherine Gebotys. Methodology for EM fault injection: Charge-based fault model. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 256–259. IEEE, 2019.

[78] Linaro Ltd. Trusted Firmware Open Governance Project, 2020. https://www.trustedfirmware.org/.

[79] Jake Longo, Elke De Mulder, Dan Page, and Michael Tunstall. SoC it to EM: electromagnetic side-channel attacks on a complex system-on-chip. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 620–640. Springer, 2015.

[80] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016.

[81] Tarjei Mandt, Mathew Solnik, and David Wang. Demystifying the secure enclave processor. *Black Hat Las Vegas*, 2016. https://www.blackhat.com/docs/us-16/materials/us-16-Mandt-Demystifying-The-Secure-Enclave-Processor.pdf.

[82] Konstantinos Markantonakis, Michael Tunstall, Gerhard Hancke, Ioannis Askoxylakis, and Keith Mayes. Attacking smart card systems: Theory and practice. *Information Security Technical Report*, 14(2):46–56, 2009.

[83] Keith E Mayes and Konstantinos Markantonakis. *Smart cards, tokens, security and applications*, volume 2. Springer, 2008.

[84] Alexandre Menu, Shivam Bhasin, Jean-Max Dutertre, Jean-Baptiste Rigaud, and Jean-Luc Danger. Precise spatio-temporal electromagnetic fault injections on data transfers. In *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 1–8. IEEE, 2019.

[85] Thomas S Messerges, Ezzat A Dabbish, and Robert H Sloan. Examining smart-card security under the threat of power analysis attacks. *IEEE transactions on computers*, 51(5):541–552, 2002.

[86] Thomas S Messerges, Ezzy A Dabbish, and Robert H Sloan. Investigations of power analysis attacks on smartcards. *Smartcard*, 99:151–161, 1999.

[87] David P Montminy, Rusty O Baldwin, Michael A Temple, and Mark E Oxley. Differential electromagnetic attacks on a 32-bit microprocessor using software defined radios. *IEEE transactions on information forensics and security*, 8(12):2101–2114, 2013.

[88] Nicolas Moro, Amine Dehbaoui, Karine Heydemann, Bruno Robisson, and Emmanuelle Encrenaz. Electromagnetic fault injection: towards a fault model on a 32-bit microcontroller. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 77–88. IEEE, 2013.

[89] Mehari Msgna, Konstantinos Markantonakis, and Keith Mayes. Precise instruction-level side channel profiling of embedded processors. In *International conference on Information Security Practice and Experience*, pages 129–143. Springer, 2014.

[90] Kit Murdock, David Oswald, Flavio D Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based fault injection attacks against intel sgx. In *IEEE Symposium on Security and Privacy*, 2020.

[91] Yuto Nakano, Youssef Souissi, Robert Nguyen, Laurent Sauvage, Jean-Luc Danger, Sylvain Guilley, Shinsaku Kiyomoto, and Yutaka Miyake. A pre-processing composition for secret key recovery on Android smartphone. In *IFIP International Workshop on Information Security Theory and Practice*, pages 76–91. Springer, 2014.

[92] NCC Group. There's A Hole In Your SoC: Glitching The MediaTek BootROM, 2020. https://research.nccgroup.com/2020/10/15/theres-a-hole-in-your-soc-glitching-the-mediatek-bootrom/.

[93] Sıddıka Berna Örs, Elisabeth Oswald, and Bart Preneel. Power-analysis attacks on an FPGA—first experimental results. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 35–50. Springer, 2003.

[94] Dan Page and Frederik Vercauteren. A fault attack on pairing-based cryptography. *IEEE Transactions on Computers*, 55(9):1075–1080, 2006.

[95] Jungmin Park, Xiaolin Xu, Yier Jin, Domenic Forte, and Mark Tehranipoor. Power-based side-channel instruction-level disassembler. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[96] Stjepan Picek, Ioannis Petros Samiotis, Jaehun Kim, Annelie Heuser, Shivam Bhasin, and Axel Legay. On the performance of convolutional neural networks for side-channel analysis. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 157–176. Springer, 2018.

[97] Vincent Pouget, Alexandre Douin, Gilles Foucard, Paul Peronnard, Dean Lewis, Pascal Fouillat, and Raoul Velazco. Dynamic testing of an SRAM-based FPGA by time-resolved laser fault injection. In *2008 14th IEEE International On-Line Testing Symposium*, pages 295–301. IEEE, 2008.

[98] Pengfei Qiu, Dongsheng Wang, Yongqiang Lyu, and Gang 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*, pages 195–209, 2019.

[99] Pengfei Qiu, Dongsheng Wang, Yongqiang Lyu, and Gang Qu. VoltJockey: Breaking SGX by software-controlled voltage-induced hardware faults. In *2019 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pages 1–6. IEEE, 2019.

[100] Qualcomm. File based encryption, 2019. https://www.qualcomm.com/media/documents/files/file-based-encryption.pdf.

[101] Qualcomm. Guard Your Data with the Qualcomm Snapdragon Mobile Platform, 2020. https://www.qualcomm.com/media/documents/files/guard-your-data-with-the-qualcomm-snapdragon-mobile-platform.pdf.

[102] Qualcomm. Inline crypto engine (ufs), 2020. https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp3124.pdf.

[103] J-J Quisquater and D Samyde. Eddy current for magnetic analysis with active sensor. *Proceedings of Esmart, 2002*, pages 185–194, 2002.

[104] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In *International Conference on Research in Smart Cards*, pages 200–210. Springer, 2001.

[105] Lionel Riviere, Zakaria Najm, Pablo Rauzy, Jean-Luc Danger, Julien Bringer, and Laurent Sauvage. High precision fault injections on the instruction cache of ARMv7-M architectures. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 62–67. IEEE, 2015.

[106] Sami Saab, Pankaj Rohatgi, and Craig Hampel. Side-channel protections for cryptographic instruction set extensions. *IACR Cryptol. ePrint Arch.*, 2016:700, 2016.

[107] Samsung. Samsung TEEGRIS, 2020. https://source.android.com/security/trusty.

[108] Samsung. What is a Knox Warranty Bit and how is it triggered?, 2020. https://docs.samsungknox.com/admin/knox-platform-for-enterprise/faqs/faq-115013562087.htm.

[109] Jörn-Marc Schmidt, Michael Hutter, and Thomas Plos. Optical fault attacks on AES: A threat in violet. In *Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 13–22. IEEE, 2009.

[110] Jörn-Marc Schmidt, Thomas Plos, Mario Kirschbaum, Michael Hutter, Marcel Medwed, and Christoph Herbst. Side-channel leakage across borders. In *International Conference on Smart Card Research and Advanced Applications*, pages 36–48. Springer, 2010.

[111] Peter Schwabe and Ko Stoffelen. All the AES you need on Cortex-M3 and M4. In *International Conference on Selected Areas in Cryptography*, pages 180–194. Springer, 2016.

[112] Carlos Segarra, Ricard Delgado-Gonzalo, Mathieu Lemay, Pierre-Louis Aublin, Peter Pietzuch, and Valerio Schiavoni. Using trusted execution environments for secure stream processing of medical data. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 91–107. Springer, 2019.

[113] Bodo Selmke, Florian Hauschild, and Johannes Obermaier. Peak clock: Fault injection into PLL-based systems via clock manipulation. In *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop*, pages 85–94, 2019.

[114] Carlton Shepherd. *Techniques for Establishing Trust in Modern Constrained Sensing Platforms with Trusted Execution Environments*. PhD thesis, Royal Holloway, University of London, 2019.

[115] Carlton Shepherd, Raja Naeem Akram, and Konstantinos Markantonakis. EmLog: tamper-resistant system logging for constrained devices with TEEs. In *IFIP International Conference on Information Security Theory and Practice*, pages 75–92. Springer, 2017.

[116] Carlton Shepherd, Raja Naeem Akram, and Konstantinos Markantonakis. Establishing mutually trusted channels for remote sensing devices with trusted execution environments. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, pages 1–10, 2017.

[117] Carlton Shepherd, Raja Naeem Akram, and Konstantinos Markantonakis. Remote credential management with mutual attestation for trusted execution environments. In *IFIP International Conference on Information Security Theory and Practice*, pages 157–173. Springer, 2018.

[118] Carlton Shepherd, Ghada Arfaoui, Iakovos Gurulian, Robert P Lee, Konstantinos Markantonakis, Raja Naeem Akram, Damien Sauveron, and Emmanuel Conchon. Secure and trusted execution: Past, present, and future-a critical review in the context of the internet of things and cyber-physical systems. In *IEEE TrustCom*, pages 168–177. IEEE, 2016.

[119] Sergei P Skorobogatov and Ross J Anderson. Optical fault induction attacks. In *International workshop on cryptographic hardware and embedded systems*, pages 2–12. Springer, 2002.

[120] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. CLKSCREW: exposing the perils of security-oblivious energy management. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1057–1074, 2017.

[121] TechInsights. Xiaomi Mi 10 Teardown Analysis, 2020. https://www.techinsights.com/blog/xiaomi-mi-10-teardown-analysis.

[122] N. Timmers, A. Spruyt, and M. Witteman. Controlling PC on ARM using fault injection. In *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 25–35, 2016.

[123] Niek Timmers and Cristofaro Mune. Escalating privileges in Linux using voltage fault injection. In *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 1–8. IEEE, 2017.

[124] Niek Timmers and Albert Spruyt. Bypassing secure boot using fault injection, 2016. https://www.blackhat.com/docs/eu-16/materials/eu-16-Timmers-Bypassing-Secure-Boot-Using-Fault-Injection.pdf.

[125] Randy Torrance and Dick James. The state-of-the-art in IC reverse engineering. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 363–381. Springer, 2009.

[126] Trustonic. Trustonic TEEs, 2020. https://www.trustonic.com/technology/.

[127] Jasper GJ Van Woudenberg, Marc F Witteman, and Federico Menarini. Practical optical fault injection on secure microcontrollers. In *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 91–99. IEEE, 2011.

[128] A. Vasselle, H. Thiebeauld, Q. Maouhoub, A. Morisset, and S. Ermeneux. Laser-induced fault injection on smartphone bypassing the secure boot. In *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 41–48, 2017.

[129] Emanuel Von Zezschwitz, Anton Koslow, Alexander De Luca, and Heinrich Hussmann. Making graphic-based authentication secure against smudge attacks. In *Proceedings of the 2013 international conference on Intelligent user interfaces*, pages 277–286, 2013.

[130] Huanyu Wang. Side-channel analysis of AES based on deep learning. Master's thesis, KTH Stockholm, 2019. https://www.diva-portal.org/smash/get/diva2:1325691/FULLTEXT01.pdf.

[131] Ruize Wang, Huanyu Wang, and Elena Dubrova. Far field EM side-channel attack on AES using deep learning. In *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security*, pages 35–44, 2020.

[132] Wired. Declassified NSA document reveals the secret history of TEMPEST, 2008. https://www.wired.com/2008/04/nsa-releases-se/.

[133] Sung-Ming Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Transactions on computers*, 49(9):967–970, 2000.

[134] YongBin Zhou and DengGuo Feng. Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing. *IACR Cryptol. ePrint Arch.*, 2005:388, 2005.