THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à l'École Normale Supérieure de Paris

Sécurité des applications sur systèmes non maîtrisés. Étude des risques, protections, enjeux et intérêts autour de la confiance dans les produits informatiques sur étagère.

Application security on uncontrolled systems. Study of the risks, protections, stakes and interests around trust in off-the-shelf computer products.

Soutenue par
Vincent Giraud
Le jeudi 26 septembre 2024

École doctorale n°386
Sciences Mathématiques de Paris Centre

Spécialité
Informatique

Composition du jury :

Pascal Lafourcade Professeur, Université Clermont Auvergne	*Rapporteur et président du jury*
Clémentine Maurice Chargée de recherche HDR, CNRS, Lille	*Rapporteuse*
Aurélien Francillon Professeur, EURECOM Sophia Antipolis	*Examinateur*
Sophie Quinton Chargée de recherche, Inria Grenoble	*Examinatrice*
Guillaume Bouffard Ingénieur de recherche, ANSSI, Paris	*Co-encadrant*
David Naccache Professeur, École Normale Supérieure de Paris	*Directeur de thèse*
Tania Richmond Maîtresse de conférences, Université de la Nouvelle-Calédonie	*Invitée*

Acronyms

AC Alternating Current 64

ACL Access-Control List 23, 24, 26

ADB Android Debug Bridge 68

ADC Analog to Digital Converter 58

AES Advanced Encryption Standard xiii, 16, 17, 20, 21, 52

AOSP Android Open Source Project 67

AOT Ahead-Of-Time 23

API Application Programming Interface 9, 63, 83

ART Android RunTime 23, 62–64, 66

ASLR Address Space Layout Randomization 22

ATM Automated Teller Machine 7, 26

AVF Android Virtualization Framework 9

BCH Bose–Chaudhuri–Hocquenghem 33

BGE Billet-Gilbert-Ech-Chatbi 20

BOM Bill Of Materials 62

BYOD Bring Your Own Device 24

CA Certification Authority 84, 85

CB Cartes Bancaires 13

COTS Commercial Off-The-Shelf ix, xi, xii, xxiii, 3, 7, 10–14, 26, 34, 56, 79, 81–85, 116

CPoC Contactless Payments on COTS xii, 12, 13

CSP Chip-Scale Packaging 60

CVM Cardholder Verification Method 12, 75

DC Direct Current 64

DCA Differential Computation Analysis 21

DES Data Encryption Standard xiii, 7, 16, 17, 20

DFA Differential Fault Analysis 20, 21

DMA Digital Markets Act 84

DPA Differential Power Analysis 21

DRM Digital Rights Management 9, 19

ECDSA Elliptic Curve Digital Signature Algorithm 22

EL Exception Level 9

EMVco Europay Mastercard Visa & Co xii, xiii, 12, 13

EOR Exclusive OR xvi, xvii, 37, 38, 40, 43, 51

eSE embedded Secure Element 10, 11, 13

eSIM Embedded Subscriber Identity Module 10

FBE File-Based Encryption 24

FDE Full-Disk Encryption 24

fTPM firmware Trusted Platform Module 9

GPU Graphics Processing Unit 4, 32, 57

GRS Generalized Reed-Solomon 33

HAL Hardware Abstraction Layer 4

HCE Host Card Emulation 10, 11, 80

HFE Hidden Field Equation 21

HMM Hidden Markov Model 71

HSM Hardware Security Module xi, xxiii, 3, 6, 7, 9, 13, 22

HTC High Tech Computer 4

IC Integrated Circuit 55, 60–62, 76, 81

IDFM Île-de-France Mobilités xii, 11

IEEE Institute of Electrical and Electronics Engineers 21

IETF Internet Engineering Task Force 6

INSA Institut National des Sciences Appliquées v

IPC Inter-Process Communication 63

ISD Information Set Decoding 31, 33

iSIM Integrated Subscriber Identity Module 10

IT Information Technology 7

JIT Just-In-Time 22, 23

JNI Java Native Interface 23

KEM Key Encapsulation Mechanism 30

MFA Multi-Factor Authentication 5

MMU Memory Management Unit 22

MPoC Mobile Payments on COTS xii, 12, 13

MSB Most Significant Bit 39, 41, 47, 49

NDA Non-Disclosure Agreement 84

NFC Near Field Communication 10, 12

NIST National Institute of Standards and Technology xv, xxi, 32, 82

OCV Open Circuit Voltage 57–59

OEM Original Equipment Manufacturer 67, 80

OMTP Open Mobile Terminal Platform 8

OS Operating System 4, 6, 8, 9, 22–24, 26, 55, 57, 60, 62–64, 67, 70, 76, 81, 84, 85

PC Personal Computer 7, 8, 83, 84

PCB Printed Circuit Board 7, 10, 60, 61, 83

PCI Express Peripheral Component Interconnect Express 7

PCI SSC Payment Card Industry Security Standards Council xii, 12, 13

PFA Persistent Fault Analysis 20

PIN Personal Identification Number 5, 12, 24, 26, 55, 67–76, 80–82

PKC Public Key Cryptosystem 32, 33

PMIC Power Management Integrated Circuit 60

POI Point Of Interaction 13

POS Point Of Sale 12

PTS PIN Transaction Security 12

PWA Progressive Web App 84

RAM Random Access Memory 79

RATP Régie Autonome des Transports Parisiens 80

REE Rich Execution Environment 6, 13, 14, 34, 76, 80, 82

RFC Request For Comments 5

RNG Random Number Generator 7

ROM Read-Only Memory 67, 68, 74, 80

RoT Root of Trust xxiii, 3, 6, 7, 14, 82, 84, 85

RSA Rivest–Shamir–Adleman xiv, xv, 21, 30–32, 51

RSB Reverse SuBstract xvi, 38, 40, 43, 51

RTC Real-Time Computing 4

SBMP Software-Based Mobile Payment xiii, 13

SDP Syndrome Decoding Problem 31

SE Secure Element xii, xiii, xxi, xxiii, 3, 6, 9–11, 13, 16, 22, 80, 82–85

SELinux Security-Enhanced Linux xviii, 23, 24, 62, 64, 85

SESAM-Vitale Système Électronique de Saisie de l'Assurance Maladie 5

SGX Software Guard Extensions 8, 9

SHA Secure Hash Algorithm 8

SIM Subscriber Identity Module xii, 9–11, 13

SIMD Single Instruction Multiple Data 32, 57

SoC System-on-Chip 9, 10, 13, 20, 60, 61, 83

SoK Systemization of Knowledge 83

SPoC Software-based PIN entry on COTS xii, xv, 12, 13, 26

SPT Software Protection Tool 13

SSL Secure Sockets Layer 10

TCB Trusted Computing Base 9

TCG Trusted Computing Group 7, 8

TDX Trust Domain Extensions 8

TEE Trusted Execution Environment xii, xxi, xxiii, 3, 6, 8, 9, 11, 13, 22, 82, 83, 85

TLS Transport Layer Security 10

TOTP Time-based One-Time Password 5

TPM Trusted Platform Module xi, xiii, xxiii, 3, 6–9, 13, 16, 82

UEFI Unified Extensible Firmware Interface 84

UI User Interface 23, 67, 69, 76

USB Universal Serial Bus 61, 64, 67

VM Virtual Machine 8, 9, 64

W3C World Wide Web Consortium 6

WAP Wireless Application Protocol 4

WLP Wafer-Level Packaging 60

Remerciements

Un doctorat est un travail de longue haleine et chaque support, qu'il soit plus ou moins académique, ou plus ou moins informel, y est précieux. Ainsi je souhaite mentionner ici ceux qui m'ont aidé à en voir le bout.

Je dois une fière chandelle à Guillaume Bouffard, qui depuis ma sortie de l'Institut National des Sciences Appliquées (INSA) aura su m'offrir des opportunités inestimables dont j'espère avoir été à la hauteur. Son support ainsi que son accompagnement, ininterrompus et si précieux, auront été primordiaux dans l'achèvement de ce travail. Je lui témoigne mon entière reconnaissance.

Je remercie David Naccache, qui, en me permettant d'entreprendre cette thèse, m'aura offert une chance que je chéris, et qui n'est jamais à court de sujets tous plus intéressants les uns que les autres.

Évidemment je souhaite exprimer ma gratitude envers tout le jury jugeant cette thèse: Pascal Lafourcade et Clémentine Maurice pour le temps et le soin qu'ils ont apportés à la revue de mon manuscrit et de mes travaux, Aurélien Francillon et Sophie Quinton pour leur participation bienvenue, et Tania Richmond d'avoir contribué à l'examination de ma thèse malgré un contexte si difficile.

J'ai une pensée pour mes collègues au sein d'Ingenico, à qui je dois un environnement de travail qui m'aura permis d'avancer dans mes recherches.

Je salue tous les copains et copines de l'association Deuxfleurs, avec qui j'apprécie toujours de redonner du sens aux technologies numériques.

Je n'oublie pas mes amis du collège, du lycée, de l'INSA, de Paris, de Rennes, de Toulouse, ainsi que mes proches et ma famille de manière générale.

Enfin je remercie Pierre Bourdieu qui sait mieux que quiconque expliquer le milieu académique parisien, le Monde diplomatique qui je l'espère n'aura pas trop déteint sur mon style rédactionnel, et Dance Dance Revolution d'imposer de sortir de chez soi.

*« Le chercheur ne peut arriver qu'après la fête, quand les lampions sont éteints et les tréteaux retirés, et avec un produit qui n'a plus aucun des charmes de l'*impromptu. *Construit contre les questions surgies de l'immédiateté de l'événement, énigmes plutôt que problèmes, appelant des prises de position totales et définitives plutôt que des analyses nécessairement partielles et révisibles, le protocole scientifique n'a pas pour lui la belle clarté du discours du bon sens à qui il n'est pas difficile d'être simple puisqu'il commence toujours par simplifier. »*

— Pierre Bourdieu [Pie84]

Résumé

Les fournisseurs de services ont de plus en plus numérisé les biens qu'ils mettent à disposition des utilisateurs, que ce soit dans le domaine de la monétique, de l'identification ou des transports, entre autres. La conséquence directe est une forte concentration des contenus sensibles sur les appareils personnels, qu'il s'agisse de données ou d'exécutables, à la fois pour la vie privée et professionnelle.

Ces systèmes dits sur étagère sont en grande partie sous le contrôle des utilisateurs finaux. Les acteurs industriels et étatiques à la recherche d'environnements stables et de confiance rencontrent de nombreux défis lorsqu'ils cherchent à y déployer des solutions, puisqu'ils ne peuvent pas nécessairement exploiter les mécanismes de sécurité matériels, s'il y en a. Certains s'orientent alors vers des implémentations visant le modèle de sécurité en boîte blanche, impliquant que toutes les opérations puissent se dérouler à la vue de tous.

Les limites générales autour de ce paradigme sont abordées dans cette thèse. D'abord, la robustesse du cryptosystème de McEliece dans un tel contexte, avec son algorithme asymétrique aujourd'hui pertinent pour différentes raisons, a été étudiée, menant à la découverte d'une attaque sur sa clé privée basée sur l'injection de faute. Ensuite, la confidentialité des processus sur les plateformes Android a été remise en question. En particulier, il est montré comment, en instrumentalisant la gestion de l'énergie sur ces systèmes, une application malveillante peut en espionner une autre. Un cas pratique visant la récupération de codes numériques secrets tapés dans d'autres contextes est explicité. Des ajustements et améliorations ont malgré tout été proposés pour ces deux sujets.

Les travaux présentés contestent la vision hégémonique autour de la sécurité dans les produits informatiques sur étagère. Ceux-ci sont sous la responsabilité des utilisateurs mais peuvent également contenir des enclaves privatives régies par le secret et faisant office de racines de confiance. La robustesse des logiciels sensibles dépend alors de paramètres inéquitables tels que l'identité des développeurs, les ressources de l'utilisateur, et les choix techniques et personnels qu'il a réalisés. Une nouvelle organisation des responsabilités liées à la sécurité est proposée.

Sécurité ⋆ Système ⋆ Confiance ⋆ Boîte blanche ⋆ McEliece ⋆ Android

Abstract

Services providers from a wide variety of domains (whether it is payments, identification, transport, or others) more and more digitalized the assets they provide to end users. In practice, this results in a strong concentration of sensitive contents (both data and executables) on people's personal smartphones, for their private and work life simultaneously.

This type of systems, often called Commercial Off-The-Shelf (COTS) devices, are under the control of end users. Industrial and state actors seeking trustable, stable execution environments thus face many challenges when trying to deploy solutions on them, as they do not necessarily have the possibility to exploit the hardware security mechanisms, if any. In consequence, they sometimes try to develop software operating in the white-box security model, where everything has to operate under open scrutiny.

In this thesis, the general limits of such a paradigm are addressed. First, it is showed how the McEliece cryptosystem, and its asymmetrical algorithm with varied advantages today, can be vulnerable in such a context by demonstrating an attack based on fault injection against it. Secondly, a focus is provided on the confidentiality guarantees provided by Android platforms. Specifically, by inspecting the implementation of power management in these, it is demonstrated how a malicious application can spy on a legitimate one. A concrete given example is focused on the recovery of Personal Identification Numbers codes entered in other software contexts. In both of these subjects, possible improvements and adjustments are nevertheless described.

These works intend to question the way security is envisioned on people's COTS devices. These are under the responsibility of end users, but can contain private parts based on secrets as roots of trust. The robustness of sensitive software thus depends on unfair parameters such as who is the developer, what can afford the end users, and what technical and personal choices has he made. A new distribution of security responsibilities is finally proposed.

Security ⋆ System ⋆ Trust ⋆ White-box ⋆ McEliece ⋆ Android

Chapter

Synthèse en français

0.1 Introduction

0.1.1 Contexte et motivation

Les systèmes embarqués sont de plus en plus répandus, et ceci à la fois dans l'industrie qu'au sein de la société. La définition associée à cette appellation n'est pas précise. Rigoureusement et historiquement, on l'utilisait pour désigner les appareils dédiés à une tâche spécifique, parfois liés à une exécution en temps réel. Ils présentaient fréquemment une faible abstraction logicielle par-dessus le matériel, se passant parfois de ce qu'on considère communément comme un système d'exploitation. Leur source d'énergie pouvait être fixe et de confiance, ou mobile et contingente. Depuis leur existence, ces appareils sont continuellement devenus plus puissants.

Cependant, l'explosion des télécommunications mobiles durant les années 1990 a remis en cause cette définition. Avec l'avènement des téléphones sans fils, les contraintes usuellement associées à l'embarqué ont glissé vers la mobilité ainsi que la gestion de l'énergie embarquée et des communications. De plus, ces systèmes ont vu leur puissance et leurs potentiels s'accroître fortement, et leurs créateurs ont rapidement développé l'ambition de permettre l'exécution et la consultation de contenus provenant de sources extérieures, tierces. L'arrivée des ordiphones, ou téléphones dits intelligents, avec leurs ressources matérielles fortes et leurs systèmes d'exploitation larges et audacieux à la fin des années 2000, constitue le parachèvement de cette évolution.

Ces appareils devenant extrêmement répandus au sein de la société civile [SAL22], ils ont dès lors automatiquement accumulé les fonctionnalités et responsabilités. N'importe quelle personne peut les utiliser à des fins de communications ou stockage chiffrés. De même, ils sont progressivement acceptés comme moyens d'identification étatique, comme titre de transport, ou comme moyens d'authentification à des services privés. L'intégration de fonctionnalités biométriques participe au développement de ces usages.

L'accumulation de ces services sensibles a nécessité la présence de mécanismes matériels de sécurité au sein des machines, afin que ceux-ci fassent office de racines de confiance. Cette appellation fait référence à des sous-systèmes ou à des fonctionnalités qui peuvent être utilisés comme un ancrage de confiance dans un système global jugé potentiellement malveillant. Elles proposent ainsi des garanties d'intégrité, de non-altérité et potentiellement d'authentification, utiles pour sécuriser des opérations sensibles dans des environnements non contrôlés. Les appareils « vendus sur étagère », qualifiés usuellement comme Commercial Off-The-Shelf (COTS), font partie de ces environnements : les ordiphones achetés en magasin finissent bien souvent entre les mains d'utilisateurs dont le comportement est imprévisible.

Les mécanismes matériels de sécurité pouvant servir comme racine de confiance sont variés. Les modules de sécurité matériel, ou Hardware Security Modules (HSMs), sont des systèmes entiers et indépendants, répandus dans l'industrie à partir des années 1990, surtout à des fins de sécurisation dans les milieux bancaires ou de l'internet. Les modules de plateforme de confiance, ou Trusted Platform Modules (TPMs), eux, prennent plus souvent, mais non obligatoirement, la forme de petits composants intégrés dans des machines ou ordinateurs. Ils répondent au standard du même nom, et fournissent ainsi un ensemble fixe de fonctions et services déterminés, liés

à la cryptographie. Ils sont notamment beaucoup utilisés pour vérifier l'intégrité de solutions logicielles avant exécution. Les environnements d'exécution de confiance, ou Trusted Execution Environments (TEEs), sont des environnements d'exécution où un programme bénéficie de nombreuses garanties basées sur la confidentialité, et l'intégrité. L'utilisation du terme abstrait « environnement » est justifiée par le fait que celui-ci peut être fourni par le même processeur qui exécute le contenu jugé comme potentiellement malveillant. La distinction s'opère alors avec l'activation de certaines fonctionnalités matérielles pour sécuriser les opérations. Enfin, les éléments sécurisés, ou Secure Elements (SEs), sont des puces indépendantes mais potentiellement intégrées au sein d'un appareil, bénéficiant de protections matérielles et logicielles afin d'à leur tour proposer un système dédié aux opérations arbitraires sensibles, qui peuvent être sollicitées via des canaux de communication. Cette dernière catégorie inclut les cartes à puce, telles que les cartes bancaires, ou celles constituant des modules d'identification de l'abonné, ou Subscriber Identity Modules (SIMs). De manière générale, les mécanismes matériels de sécurité sont particulièrement prisés et adaptés à la monétique.

Les paiements sur machines personnelles ont été possibles dès le développement de la Toile. Les utilisateurs ont très tôt pu acheter en ligne, en rentrant manuellement leurs informations de carte bancaire dans des formulaires. Si les communications de celles-ci ont rapidement été protégées grâce au chiffrement, ce n'est pas le cas de leur manipulation en local. Au milieu des années 2010, lorsque le paiement par mobile a fait son apparition, la protection des données bancaires au sein des appareils a faite l'objet de mesures plus strictes. Lorsqu'un usager intègre sa carte bancaire au sein de son ordiphone, les informations de celle-ci en elles-mêmes n'y restent pas. En lieu et place, un jeton correspondant est généré, conservé en confidentialité, puis utilisé à terme. Cette indirection permet une réduction des risques, tout en offrant les mêmes fonctionnalités.

Payer en utilisant son ordiphone est une fonctionnalité notamment proposée par les entreprises Apple, Samsung et Google. Le fait qu'elles soient également constructeurs de téléphones intelligents n'est pas une coïncidence. En effet, les mécanismes matériels de sécurité pouvant opérer les opérations nécessaires ne sont pas accessibles par tout le monde : seuls les manufacturiers peuvent pleinement les exploiter. Un exemple emblématique d'acteur ayant subi cette restriction d'accès est l'établissement Île-de-France Mobilités (IDFM), qui, pendant longtemps, n'a pu implémenter le titre de transport numérisé sur ordiphone que grâce au bon vouloir de l'opérateur téléphonique Orange, autorisant l'exploitation de leurs cartes à ces fins. Dans le cadre de la monétique, ces contraintes d'accès pénalisent tout autant l'émission que la réception de paiements sur ordiphone.

La monétique est un domaine industriel reposant fortement sur l'autorégulation. Depuis 1995 ses acteurs se plient aux exigences de l'organisme Europay Mastercard Visa & Co (EMVco), et à celles du conseil des standards de sécurité de l'industrie du paiement par carte, ou Payment Card Industry Security Standards Council (PCI SSC), depuis 2006. Ces deux entités ont substantiellement participé à l'évolution et à la direction de ce domaine, l'arrivé du paiement sur mobile ne faisant pas figure d'exception. PCI SSC a en effet émis trois standards absolument constitutifs de cette technologie: Software-based PIN entry on COTS (SPoC) en 2018, pour régir uniquement la prise en charge d'entrées de codes secrets dans des produits sur étagère, Contactless Payments on COTS (CPoC) en 2019 pour réglementer l'acceptation de paiements sans contact et sans authentification du propriétaire de la carte sur cette classe d'appareils, et Mobile Payments on COTS (MPoC) en 2022, pour sécuriser et encadrer de manière générale toute acception sur les systèmes dits COTS.

Pour les acteurs tiers, c'est-à-dire ceux n'étant pas des constructeurs d'appareils vendus sur étagère ou de mécanismes matériels de sécurité, opérer sûrement un paiement dans de tels environnements s'apparente à un véritable défi dont l'accomplissement est incertain. N'ayant accès à aucune des deux racines de confiance fréquemment présentes dans les ordiphones ou les tablettes (les SEs et les TEEs), ils sont contraints d'opérer dans un contexte sur lequel

ils n'ont aucun pouvoir de supervision. Cette restriction pose une profonde inégalité parmi les acteurs de l'industrie du paiement, et peut également poser des questions de souveraineté nationale, y compris pour un pays comme la France, pourtant historiquement meneur dans la monétique. Face à cet enjeux, un domaine de recherche apparaît comme tout à fait judicieux : l'exploration du modèle de sécurité en boîte blanche, qui implique un contexte d'exécution où tout doit pouvoir se dérouler sous les yeux d'acteurs potentiellement malveillants. Ce paradigme a dépassé le milieu de la recherche académique, et est notamment reconnu officiellement par EMVco dans son standard de paiement mobile basé sur le logiciel, ou Software-Based Mobile Payment (SBMP).

0.1.2 État de l'art

Lorsqu'on envisage l'implémentation de contenus ou programmes sensibles, le modèle de sécurité le plus évident est assimilé à une boîte noire : un système où tout se concentre au sein d'un environnement fermé, empêchant d'inspecter le fonctionnement interne, d'en récupérer une quelconque information, ou de l'influencer. Le seul levier d'action disponible est les entrées de l'algorithme, et la seule lecture possible se situera aux sorties de celui-ci. Les TPMs et les SEs sont des exemples visant à atteindre ce modèle. Il convient cependant de noter que leurs communications extérieures, elles, sont lisibles par tous. D'où, bien souvent, leur chiffrement, lorsque c'est possible.

En 2002 est introduit le modèle de sécurité dit boîte blanche, avec en guise de démonstration, des implémentations du standard de chiffrement avancé, ou Advanced Encryption Standard (AES) [CEJP03b], et du standard de chiffrement des données, ou Data Encryption Standard (DES) [CEJP03a]. L'objectif visé est ainsi de pouvoir conserver et exécuter de tels programmes dans des environnements ouverts, auditables par tous, sans pouvoir compter sur une quelconque confidentialité, que ce soit durant les opérations ou non. Cette ambition implique notamment de prévoir des inspections avancées et des instrumentations par des acteurs malveillants, le tout sans compromettre une potentielle clé secrète dans le cas d'une implémentation cryptographique.

Pour approcher cet objectif, la mesure la plus exploitée est la précalculation de résultats intermédiaires. Au lieu de réaliser réellement toutes les opérations mathématiques nécessaires à l'implémentation d'un algorithme, celles-ci verront tous leurs résultats possibles déterminés à l'avance en fonction de toutes les entrées possibles, puis conservés dans des tables au sein de l'exécutable. Cette technique requiert bien souvent d'intervenir sur l'organisation des calculs, afin de limiter le nombre d'entrées possibles pour chacun d'entre eux. En effet, le nombre de résultats est exponentiel par rapport à la taille des entrées; en l'absence de précautions, les opérations considérées seront fréquemment impossibles à précalculer, de par le poids des potentielles tables associées.

Ces tables, dont les valeurs peuvent dépendre de secrets, ne sont pas sécurisées telles quelles. Dans l'AES par exemple, celles associées à l'opération `AddRoundKey` permettent de récupérer aisément la clé si rien n'est fait. Pour y remédier, deux mesures sont mises en place. La première est celle de fusionner les opérations considérées avec des encodages. Il s'agit de bijections générées au hasard, destinées à augmenter la confusion au sein de l'implémentation. En en appliquant une à la fin d'une opération, puis l'exact inverse au début de la suivante, les tables sont impactées mais le résultat des deux calculs associés reste le même. En déployant parcimonieusement cette logique à l'intégralité d'un algorithme, on conserve sa sémantique globale. La deuxième mesure est la fusion, de la même manière, de transformations linéaires. Également sélectionnée aléatoirement, elles doivent cependant être inversibles. Cette fois-ci, leur rôle est d'augmenter la diffusion. Ces mesures, si elles peuvent compliquer fortement l'inspection par un attaquant, protègent malgré tout les tables localement. L'analyse des interactions de données au niveau de l'algorithme global, ainsi que des vices de conception au sein de la spécification visée, peuvent porter leurs fruits dans le cadre d'un travail malveillant.

Il est fréquent que les attentes autour des implémentations en boîte blanche soient confuses. Un risque particulièrement ignoré est celui de la copie de code : des fois, il n'est pas nécessaire d'investir le moindre effort dans l'étude malveillante d'un exécutable contenant un secret, si celui-ci peut être simplement copié autre part et réutilisé tel quel. De manière formelle, on décrit les attentes autour des implémentations en boîte blanche avec trois éléments précis : l'exécution en sens unique, l'incompressibilité, et la traçabilité [DLPR13].

Il est reconnu, notamment au sein des communautés de recherche en cryptographie, que les implémentations en boîte blanche sont moins sécurisées que celles opérant dans un scénario plus conventionnel, comme en boîte noire [CP16]. Ce dernier permet d'obtenir des robustesses sur le très long terme, là où le premier impose de mesurer la sécurité d'une implémentation en semaines dans les meilleurs cas, voir en jours, ou même en heures [BDG+22, BBD+22]. Cet état de fait est principalement causé par la puissance des attaques développées à l'encontre des implémentations en boîte blanche, qu'elles soient basées sur la cryptanalyse, l'étude des canaux auxiliaires, ou l'injection de faute. Un atout déterminant qu'elles peuvent démontrer est le court-circuitage de tout le travail normalement nécessaire pour comprendre une implémentation, requérant habituellement un fort savoir-faire, ainsi que beaucoup de temps, et potentiellement de financement.

L'application du modèle boîte blanche dépasse les algorithmes de chiffrement symétriques de blocs. Développer de telles implémentations d'algorithmes asymétriques, en se concentrant sur la clé privée, est un sujet particulièrement scruté. L'algorithme RSA est difficilement portable dans ce paradigme, de par la taille de ses variables intermédiaires. Ceux basés sur les courbes elliptiques peuvent se prêter à cet exercice mais les résultats observés dans le monde académique et industriel ne sont actuellement pas satisfaisants.

Un système d'exploitation fréquemment visé par les développeurs d'exécutables en boîte blanche, car extrêmement répandu sur un parc hétérogène de machines, est Android. Développé et dévoilé durant les années 2000, basé sur le noyau Linux, il est particulièrement adapté aux ordiphones et tablettes. Étudier ses abstractions logicielles est pertinent dans la mesure où celles-ci, lorsque mal conçues, peuvent jouer à l'encontre de la sécurité d'un système.

Si Android peut exécuter du contenu provenant de n'importe quelle source, Google déploie un certain nombre d'efforts pour contrôler la fourniture d'applications via son magasin numérique officiel. L'objectif est de limiter la diffusion de celles jugées indésirables, principalement selon trois critères : la dangerosité envers le système, la dangerosité envers l'utilisateur, et l'inadéquation à certaines normes morales ou culturelles. Ce dernier n'est pas considéré dans le cadre de ce travail.

Les risques techniques que peuvent causer une application sont multiples, et répartis sur différentes couches d'abstraction. Les contre-mesures le sont tout autant. Les protections contre les accès illégaux dans les systèmes de fichiers se situent au niveau du noyau Linux. Lorsque ce genre d'accès concerne certaines ressources matérielles ou du système d'exploitation, ils sont la plupart du temps interceptés au niveau du cadriciel Android, développé surtout avec le langage Java. Certains abus liés à la vie privée ou aux ressources disponibles sur les appareils peuvent être détectés avant exécution des applications, alors qu'elles sont mises en ligne sur le magasin numérique. Ce processus d'inspection est nécessairement automatique au vu du trafic de contenus téléversés. Le chiffrement du stockage, s'il les protège efficacement les données au repos, n'est pas pertinent à l'exécution, contre les accès illégaux. Tous les risques cités ne sont pas nécessairement exclusifs, et peuvent au contraire se cumuler dans le cadre d'une attaque.

Si l'on considère l'isolation des contenus et des applications sur Android, on peut trouver de nombreux travaux tendant à l'améliorer, et proposant des mesures variées, qu'elles soient matérielles ou logicielles. Parmi les risques qui peuvent apparaître si les contre-mesures ne sont pas suffisantes, on peut compter l'espionnage de codes secrets entrés par l'utilisateur. Ces attaques sont essentiellement basées sur l'exploitation de canaux auxiliaires tels que le son, les mouvements, l'énergie, le temps, ou un mélange d'entre eux. La compromission des entrées de

codes secrets dans les systèmes basés sur Android peut contrecarrer certaines de leurs ambitions, comme celle d'opérer des transactions bancaires, attestée par le standard SPoC.

0.2 Contribution

0.2.1 Offensive sur le cryptosystème de McEliece et propositions pour implémentation en boîte blanche

Robert J. McEliece a dévoilé son cryptosystème éponyme en 1978 [McE78], une année après la publication de l'algorithme Rivest–Shamir–Adleman (RSA) [RSA78]. Son principal apport est la création d'une fonction à trappe en se basant sur les codes correcteurs d'erreurs. L'algorithme repose en grande partie sur des opérations linéaires. Il utilise les codes de Goppa [Val70] sans forcément les imposer. Aujourd'hui encore, il préserve une image positive, notamment de par sa robustesse attestée sur plusieurs décennies, et ses bonnes performances. Une de ses qualités s'est révélée relativement récemment : il propose une bonne résistance face aux attaques qui peuvent être menées par des ordinateurs quantiques. L'argument ayant favorisé son concurrent RSA à l'époque, à savoir la taille des clés, est moins contraignant aujourd'hui, de par l'évolution du matériel informatique.

Il s'agit d'un algorithme cryptographique asymétrique, où l'on chiffre avec la clé publique, et déchiffre avec la clé privée. Il est donc judicieux de l'utiliser par exemple dans le cadre d'encapsulations de clés. Pour générer une paire de clé, on commence par la partie privée, en créant une matrice de permutation P et une matrice de brouillage S, toutes deux aléatoires et inversibles. On génère ensuite un code correcteur de Goppa et sa matrice de génération G associée. On utilise des paramètres tels que k, qui est la taille des blocs avant chiffrement, n, leur taille après chiffrement, et t, le nombre maximal d'erreurs que le code sélectionné peut détecter et corriger. Au final, les matrices P, G et S séparées et distinctes représentent la clé privée, et la matrice $G' = SGP$ est la clé publique. En appliquant cette dernière à un bloc, puis en ajoutant volontairement des erreurs, seules les personnes disposant des trois matrices non fusionnées peuvent espérer retrouver le message original.

Cet algorithme fonctionne puisqu'une permutation conserve le même poids de Hamming. Ainsi, qu'importe si on applique des erreurs avant ou après une telle opération : leur nombre reste le même. Cependant, tous les acteurs qui ne peuvent réaliser juste une dépermutation seulement n'auront à leur disposition qu'une valeur, issue d'un code de Goppa non identifié. Les possibilités restantes à un attaquant, à savoir la recherche exhaustive, la recherche du plus proche voisin, ou le décodage par ensemble d'information, sont toutes disqualifiées dès lors que des paramètres assez grands ont été sélectionnés par le concepteur.

Ce cryptosystème a un fort héritage : il a été adapté, avec la variante de Niederreiter, afin de pouvoir émettre des signatures [CFS01]. Il a été l'objet de recommandations officielles pour se prémunir contre les attaques d'ordinateurs quantiques [DWTN15], et une de ses dérivations a été candidate au concours de l'institut national des standards et technologies États-Unien, ou National Institute of Standards and Technology (NIST), jusqu'à la dernière phase [Inf22]. Le débat sur les paramètres à sélectionner lorsque l'on implémente ce cryptosystème est toujours d'actualité. Un fort travail offensif a été développé à son encontre, portant ses fruits essentiellement face à des tentatives d'utiliser d'autres codes correcteurs que ceux de Goppa. Plusieurs attaques basées sur des canaux auxiliaires existent. Cependant, aucune basée sur l'injection de faute n'avait permis d'obtenir des informations sur la clé privée dans la version originale de l'algorithme. Les avantages de ce cryptosystème, combinés à la facilité de simuler logiciellement des injections de faute dans des environnements ouverts, ont justifié l'intérêt d'investiguer autour d'une telle possibilité.

L'absence de tentatives fructueuses indique que le cryptosystème de McEliece fait preuve d'une excellente résistance contre les attaques en faute. Son utilisation des codes correcteurs

y est déterminante : lorsqu'injectée naïvement, une faute est bien souvent corrigée durant le déchiffrement, comme si rien d'anormal ne s'était passé. A contrario, une trop forte corruption, si elle peut outrepasser la correction d'erreurs, résultera cependant avec des informations inutilisables. Face à ce constat, il paraît plus judicieux d'investiguer autour des implémentations plutôt que de la spécification. En effet, elles peuvent apporter des comportements exploitables par des acteurs malveillants. En particulier, on s'intéressera surtout à l'exploitation de la matrice de permutation : obtenir des informations sur celle-ci peut mettre la clé privée, dans sa globalité, en danger critique. La matrice de brouillage, même si récupérée entièrement, ne fait pas de même. Le code correcteur est difficilement attaquable car positionné entre la permutation et le brouillage, de plus, les différentes manières de le conserver en mémoire sont variées et imprévisibles.

La matrice de permutation a des propriétés utiles que l'on peut exploiter : elle n'a qu'un seul bit à 1 sur chaque ligne ainsi que chaque colonne, et doit préserver le poids de Hamming des opérandes qu'elle traite. Souvent, durant le déchiffrement, un attaquant ne peut pas lire le résultat de la dépermutation avant le décodage. Tout l'enjeu d'une attaque est alors de trouver une primitive permettant de faire passer des bribes d'information à travers le décodage et le débrouillage.

En pratique, on va considérer la manière la plus commune de faire des produits vecteur-matrice sur $\mathbb{F}_2$: faire une première boucle qui itère sur le vecteur, et qui, uniquement quand le bit considéré est à 1, lance une seconde boucle, qui va appliquer tous les OU exclusifs nécessaires entre la ligne de la matrice correspondante, et le vecteur servant d'accumulateur. Dans les corps de Galois binaires, il n'est pas possible de faire cela en utilisant une instruction d'addition classique, prévue pour l'arithmétique non modulaire. Ainsi sur une architecture Arm, on est certains que l'instruction Exclusive OR (EOR) sera utilisée, car c'est elle qui peut réaliser les OU exclusifs. En étudiant la table des codes d'opération disponibles dans ce jeu d'instruction, on constate qu'en ne basculant qu'un seul bit, on peut transformer ce EOR en Reverse SuBstract (RSB), c'est-à-dire en soustraction inversée. On se restreindra à cette manipulation, afin de considérer le scénario de faute le plus contraignant pour un attaquant.

Avec cette corruption, la suite de l'offensive repose sur une mission spécifique : générer des surcharges du code correcteur dont on peut tirer des conclusions sur la matrice de permutation. En effet, l'application de la dépermutation ne respecte ici plus la conservation du poids de Hamming. En exploitant la propriété qui veut qu'avec les codes linéaires, un mot avec un poids de Hamming inférieur à t se réfère forcément à un vecteur nul, on peut viser l'obtention d'indications sur les colonnes où se situent chaque 1 de la matrice de permutation. En effet, on constate qu'en faisant des soustractions avec deux valeurs contenant uniquement un seul 1, on peut obtenir des poids de Hamming proportionnels à la distance entre ces 1, ou à celle avec le bit de poids fort. En pratique, on va exploiter le fait que l'addition sur $\mathbb{F}_2$ entre les lignes de la matrice de permutation et le vecteur accumulateur vont se faire variables par variables, donc groupes de colonnes par groupes de colonnes. Sur Arm, le plus probable est de faire ces additions par groupes de 32 bits. Ainsi, en envoyant des entrées avec un nombre contrôlé de bits à 1, on peut utiliser diverses logiques indiquant que s'il y a surcharge du code correcteur, alors les lignes correspondantes de la matrice de permutation ont forcément leur 1 dans différents groupes de colonnes, ou au contraire forcément dans le même, selon la situation, qui dépend entre autres du sens de la soustraction générée avec la faute injectée. Itérer cette martingale en boucle permet d'accumuler petit à petit les informations sur la matrice de permutation.

Deux exemples pas-à-pas sont donnés et détaillés, afin de présenter les diverses situations auxquelles peuvent faire face des attaquants. Cette versatilité des méthodes à suivre et des résultats obtenus rendent la mesure de l'efficacité complexe, mais de manière générale, selon les paramètres sélectionnés du cryptosystème de McEliece, et selon la taille des variables utilisées normalement pour faire l'addition sur $\mathbb{F}_2$, on peut viser une réduction d'entropie de moitié voir de trois quarts sur la matrice de permutation.

Puisque simuler automatiquement une injection de faute sur une telle implémentation peut facilement s'envisager, notamment en visant spécifiquement les instructions EOR malgré une quelconque obfuscation, chercher comment l'utilisation du cryptosystème de McEliece dans un scénario en boîte blanche peut se faire est judicieux. Les multiplications de matrices, mêmes si elles sont de grandes tailles, peuvent se décomposer en produits bien plus petits, pour rendre la précalculation abordable. Une opération résiste cependant aux calculs anticipés : le décodage, qui implique une forte diffusion des données. Ainsi, pour rendre possible une version de cette algorithme destinée au scénario de la boîte blanche, envisager plusieurs sous-codes en parallèles peut être une solution. Que ce soit lors du chiffrement ou du déchiffrement, chaque sous-code devra avoir sa sous-permutation spécifique. Cependant le brouillage et débrouillage pourront être commun à tous, afin d'apporter une diffusion et répartition globales des données. Pour protéger la clé privée, une couche de confusion devrait être appliquée entre le décodage et le débrouillage, via l'insertion de bijections et transformations linéaires aléatoires. Au final, le paramètre le plus critique pour pouvoir envisager une exploitation sécurisée d'une telle implémentation est la taille des sous-codes et sous-permutations, qui représente un compromis entre difficulté pour l'attaquant, et taille de l'exécutable.

Cet axe de travail a donné lieu à des interventions à la *Journée thématique sur les Attaques par Injection de Fautes* (JAIF) 2022, et à *Security of Software/Hardware Interfaces* (SILM) 2023 au sein d'*European Symposium on Security and Privacy* (EuroS&P). De même, il a mené à la demande de brevet numéro 2210700, le 17 octobre 2022, auprès de l'*Institut National de la Propriété Industrielle* (INPI).

0.2.2 Confidentialité des processus sur Android

La capacité d'influencer d'autres processus, tout comme celle d'y récupérer des informations, peut être liée aux interactions laissées possibles avec les couches basses d'abstraction du système, au point de toucher à des enjeux matériels.

La gestion de l'énergie sur les systèmes portatifs est primordiale. Concernant les produits informatiques grand public sur étagère, l'objectif qui prime, après la sûreté, est principalement l'autonomie de la batterie. Pour l'augmenter, seules deux pistes sont exploitables : augmenter l'énergie embarquée, ou réduire la consommation. La première a d'ores et déjà été poussée aux maximums autorisés par la physique et les mesures de sûreté, et implique un temps de chargement plus long. La seconde, elle, a l'avantage d'engendrer des plateformes plus économes, donc chauffant moins, et étant plus écologiques, sans modifier le temps de chargement. Cette méthode est réalisable matériellement, mais les processus industriels le permettant sont complexes et difficilement améliorables. On peut aussi le considérer via une approche logicielle, mais cela nécessite souvent un temps et une complexité de développement plus longs, avec une collaboration entre les différentes couches d'abstraction [Nal22].

Depuis plusieurs décennies, les batteries à base de lithium représentent l'option la plus évidente pour l'alimentation des systèmes mobiles, de par leurs caractéristiques physiques. Cependant, elles sont difficiles à maîtriser. De plus, évaluer et prévoir leurs différentes caractéristiques n'est pas aisé. Cela est notamment dû à des difficultés liées à la réalisation des mesures, mais également à l'imprévisibilité de ce type de batteries. Par exemple, leur tension n'exhibe pas de décroissance continue lors de la décharge, mais trace à la place une courbe irrégulière, composée à la fois de fortes et faibles variations. Suivre la consommation et estimer son futur, en particulier sur des systèmes permettant de nombreuses activités diverses et variées, est un véritable défi. Il est également important de noter que de nombreux paramètres influencent la batterie et ses métriques, incluant par exemple l'âge, la température, et la qualité des matériaux ainsi que de la fabrication. Les enjeux autour de la maîtrise des batteries à base de lithium en conditions réelles sont d'autant plus importants qu'elles peuvent être particulièrement dangereuses si mal exploitées.

Ainsi, l'exploitation de batteries à base de lithium et sa complexité peut faire naître, auprès des concepteurs de systèmes portatifs, la tentation de déléguer cette tâche. Celle-ci est la raison d'être des jauges de carburant. Contrairement à ce que laisse entendre leur nom, ce sont des circuits intégrés indépendants, qui peuvent être rajoutés sur le circuit imprimé d'une plateforme, et qui sont en charge de réaliser des mesures précises sur la batterie électrique et l'alimentation en énergie, ainsi que de faire des estimations à partir de ces valeurs. Particulièrement pratiques dans l'exploitation d'ordiphones, tablettes et consoles portables, ces composants sont devenus de plus en plus communs en leur sein, surtout dans les produits haut de gamme, car ils sont relativement chers. Ils aident à prendre les nombreuses décisions qui peuvent être mises en place pour optimiser l'autonomie, la sécurité, et l'état de la batterie sur le long terme. L'efficacité et la précision de ces circuits intégrés, remarquablement bonne, a été vérifiée dans la littérature académique [BXZ+17].

Ces composants sont petits et sont soudés au sein même de la plateforme, durant la manufacture, préférablement proche des sources d'énergie. Conceptuellement, ils sont entre elles et le système central, de manière à ce que tout courant passe par eux, afin qu'ils puissent le mesurer dans le cadre du suivi énergétique. Un lien série permet au système central de communiquer avec, essentiellement pour pouvoir demander des valeurs ou estimations. La présence ou non d'une jauge à carburant n'étant pas inscrite sur les fiches techniques des produits sur étagère, impossible de la connaître à l'achat. C'est après que l'on peut visuellement inspecter le circuit imprimé pour en chercher une, ou, sur les plateformes basées sur Android ou Linux plus généralement, sonder logiciellement la liste des composants liés à la puissance, pour le savoir.

L'intégration et la prise en compte de ces composants par Android implique rapidement de retoucher au sujet de l'isolation autour des contenus dans ce système d'exploitation. En effet, les interactions entre applications ont toujours été fortement réglementées et limitées selon des règles claires et fixes dans le temps. Cependant, les interactions entre une application et les ressources au sein des couches d'abstraction inférieures, voir avec le matériel, le sont bien moins. Les politiques de sécurité à leur encontre dépendent des versions d'Android, de la ressource concernée, du type d'accès, et des fois même du constructeur de la plateforme. Ainsi quand les développeurs d'une application prévoient qu'elle accède à une ressource du système, la possibilité réelle de le faire est souvent découverte au moment même de l'exécution. Dès lors, savoir qui peut faire quoi avec les jauges de carburant au sein d'Android nécessite clarification.

Dès les débuts d'Android, un service système nommé `BatteryManager` a été présent. À l'époque cependant, il ne permettait que d'obtenir des indications sur la batterie, pour savoir par exemple si elle fonctionnait correctement ou était morte, ou si une source extérieure était branchée. C'est à partir de la version 5 que les applications peuvent demander des mesures, concernant par exemple la capacité restante, ou le courant consommé en temps réel, entre autres. L'exploitation et la récupération de telles données en arrière plan n'ont été modérées qu'à partir de la version 9, mais en instaurant une permission donnée instantanément sous condition de faire apparaître une notification, potentiellement mensongère. Avec l'arrivée de la version 12, une limite de fréquence des accès est imposée. Mais celle-ci, implémentée à l'aide d'une liste noire, ne concerne pas les jauges de carburant. Au final, si l'on cherche à déterminer si un quelconque acteur au sein d'Android modère les accès des applications aux jauges de carburant, on découvre qu'il n'y en a aucun : ni au niveau du cadriciel Java, ni au niveau de la machine virtuelle, ni dans les exécutables natifs, et pas non plus au sein du noyau malgré la présence de Security-Enhanced Linux (SELinux).

Face à cette réalité, on identifie trois risques créés : celui de voir l'utilisateur espionné, où une application pourrait, sur le long terme, faire un suivi énergétique pour en déduire des informations sur l'utilisation de la plateforme, ce qui nuirait à la protection de la vie privée. Un second risque est la création d'un canal de communication caché. Toutes les applications ont de facto un droit d'écriture sur le signal de consommation en temps réel, puisqu'elles peuvent l'influencer avec des traitements ou non. Elles ont également un droit en lecture comme évoqué

précédemment. Dès lors, deux applications malveillantes peuvent tenter de communiquer pour échanger des informations de manière secrète sur la plateforme, impliquant potentiellement des fuites de données contournant des manques de permissions. Enfin, un troisième risque est tout simplement celui de l'espionnage, par une application malveillante, d'une application légitime.

Pour illustrer ce troisième risque, on se concentre sur un cas pratique spécifique : l'espionnage de l'entrée de code secret numérique par l'utilisateur. Ce scénario est des plus judicieux d'abord parce que les jauges de carburant n'ont pas une fréquence de rafraîchissement suffisante pour étudier des processus à vitesse logicielle, il faut donc se limiter à ceux à vitesse humaine. Ensuite puisque les codes numériques sont fréquemment utilisés, que ce soit par Android lui-même pour le déverrouillage, ou par des applications tierces sensibles, et que leur fuite implique la plupart du temps des conséquences lourdes.

Les versions d'Android considérées dans ces travaux sont réparties entre la 9 et la 12. Des images à la fois minimales ou provenant des constructeurs ont été utilisées. Les tests ont particulièrement visé les ordiphones de la gamme Nexus et Pixel de Google, avec et sans alimentation branchée. Une application cible naïve a été créée, qui a simplement recours au pavé numérique natif d'Android. Une autre, dédiée à l'attaque, embarque le nécessaire pour pouvoir scruter les relevés de consommation de la jauge de carburant en temps réel, y compris en arrière plan. Elle affiche les résultats à l'écran dès le processus interrompu.

Le premier constat est la confirmation que les mesures des jauges de carburant sont précises au point de pouvoir détecter le toucher d'un doigt sur l'écran. Pour des raisons mélangeant vraisemblablement des phénomènes physiques et logiciels, la consommation augmente lorsque l'utilisateur appuie sur l'affichage. Cet état de fait est significatif, puisqu'il autorise à passer à la seconde étape, l'espionnage lors de l'entrée d'un code numérique. En effet, on constate que sur le signal de consommation pendant un tel processus, on voit aisément les pics associés à chaque pression. Ce fait est accentué lorsque le vibreur s'enclenche aux touchers, comme c'est souvent le cas par défaut. Ainsi, un acteur malveillant peut déterminer les temps écoulés entre chaque chiffre composant le code numérique secret.

Vient alors ensuite la phase d'exploitation des données récoltées : comment en déduire les codes les plus probables ? La littérature présente des méthodes basées sur les modèles de Markov cachés, ou sur l'apprentissage automatique. Ici, le développement déterministe d'un arbre des séquences possibles a été retenu. En effet, l'attaquant bénéficie d'une garantie importante : il sait que l'utilisateur va forcément valider son code avec la touche de confirmation, qui se situe à un endroit précis et connu. Il peut alors, en partant de la fin, déterminer à chaque fois la ou les touches précédentes les plus probables. Lorsqu'il y en a deux à équidistance, on ouvre un embranchement. En réalité, on se basera plus sur le ratio entre les 4 délais présents lorsque l'on tape un code à 4 chiffres, sans oublier la validation à la fin. Enfin, les données gyroscopiques, aussi accessibles librement dans une certaine mesure, peuvent aider à réduire l'ensemble obtenu des séquences possibles.

Ce risque impacte directement la confidentialité des systèmes basés sur Android. Le danger est important pour les plateformes concernées puisqu'il est fréquent, en résultat, d'obtenir moins de 10 possibilités, voir moins de 5, pour un code à 4 chiffres. Les potentielles contre-mesures envisageables se situent dans différentes couches d'abstraction. Parmi les plus judicieuses, on trouve au niveau applicatif l'utilisation de pavés numériques aléatoires. À la portée des développeurs tiers, celle-ci endommage cependant l'accessibilité et la réutilisation de code et services. Brouiller le signal de consommation, voir simuler volontairement des faux signaux d'entrée de code est complexe. Au niveau du système, l'implémentation d'interface utilisateur de confiance l'est également [FPP23], sans forcément donner de bons résultats puisque les sur-consommations énergétiques lors des touchers auront toujours lieu. La solution à la racine, à savoir corriger le système Android, est simple à réaliser, cependant elle n'est, elle aussi, pas accessible aux développeurs tiers, et risque de générer des incompatibilités avec les applications basée sur l'ancien fonctionnement du service `BatteryManager`.

Cet axe de travail a donné lieu à des interventions au *Symposium sur la Sécurité des Technologies de l'Information et des Communications* (SSTIC) 2023, et à *International Workshop on Security* (IWSEC) 2023 ainsi qu'à la publication associée dans *Lecture Notes in Computer Science* (LNCS) volume 14128. De même, il a mené à la demande de brevet numéro 2212834, le 6 décembre 2022, auprès de l'*Institut National de la Propriété Industrielle* (INPI).

0.3 Conclusion

0.3.1 Leçons

En une décennie, le taux d'équipement en ordiphones de la population des pays développés et la puissance de ces systèmes ont fortement augmenté. Ils ont atteint un point où, pour bien des États et des entreprises, considérer que l'intégralité de la population a un ordiphone puissant et disponible dans sa poche n'est plus un tabou. Que cela soit dans le milieu du paiement, de l'identification, ou des transports, les alternatives uniquement numériques sont intensivement poussées.

Cette évolution représente un profond changement de paradigme. Là où les cartes à puce étaient financées par les fournisseurs de services et étaient sous leur responsabilité, ces charges sont désormais attribuées aux utilisateurs finaux. Leur capacité ou non de s'offrir un ordiphone avec les bons mécanismes de sécurité matériels, et les choix techniques qu'ils feront à son encontre, comme par exemple débloquer les droits administrateurs ou réinstaller un système d'exploitation, sont significatifs et déterminants. Certains citoyens bénéficieront inévitablement de meilleures garanties de sécurité ou d'avantage de fonctionnalités par rapport à d'autres. Le modèle de sécurité en boîte blanche peut lutter contre ces inégalités en réduisant les considérations liées au matériel et à l'environnement logiciel. Cependant, qu'il soit abordé via des solutions d'obfuscation clé en main, des techniques mathématiques cryptographiques, ou dès la conception des solutions, les implémentations échouent à proposer des solutions sécurisées sur le long terme. L'effort de l'industrie envers cette piste est malgré tout toujours massif, puisque l'impossibilité d'accéder aux mécanismes de sécurité matériels et l'espoir de pouvoir déployer sur la flotte mondiale d'appareils sans effort de compatibilité la rende fortement attrayante. En pratique, elle compense l'éphémérité de la sécurité dans ce modèle avec des mécanismes de rotations régulières des secrets. Ainsi, les notions de temps et d'efforts nécessaires pour compromettre une implémentation deviennent primordiales.

Les travaux réalisés durant cette thèse exposent deux menaces différentes auxquelles s'exposent les implémentations en boîte blanche. La première est celle des attaques permettant d'éviter lesdits temps et les efforts normalement nécessaires. En effet, cela peut-être le cas de celles basées sur la simulation d'injections de fautes. Ici, la perturbation provoquée sur une manière classique d'implémenter le cryptosystème de McEliece permet de ne pas avoir à se soucier d'une quelconque obfuscation des variables intermédiaires ou du flot de contrôle. La deuxième menace est celle qui fait fi des garanties fournies par le logiciel : la promesse d'être sur une image officielle d'Android, à jour, sans droits administrateurs débloqués peut rassurer les concepteurs d'implémentations en boîte blanche. Il n'en reste pas moins qu'en ayant accès aux métriques fournies par la jauge de carburant, l'espionnage malveillant peut être une réelle possibilité. Au-delà de ces deux menaces, des propositions de contre-mesures ou d'améliorations ont tout de même été proposées.

Ces résultats confortent la vision du modèle de sécurité en boîte blanche comme un mode d'exploitation de solutions basé sur la sécurité éphémère constamment renouvelée, en course constante contre les efforts des attaquants. Ils démontrent également la difficulté d'éteindre la sûreté d'une racine de confiance vers un environnement riche et ouvert : si simuler la faute nécessaire pour l'attaque du cryptosystème de McEliece peut être impossible dans un contexte logiciel dont l'intégrité a été vérifié, espionner les autres contenus à l'aide des métriques énergétiques

resterait possible. En effet, la vérification d'intégrité d'un environnement peut le déduire comme valide même lorsqu'il contient, par voie officielle, une politique de sécurité néfaste. Au final, les travaux présentés ici rappellent que la sécurité des systèmes d'information est avant tout une chaîne constituée de maillons tous sensibles : sécurité des spécifications, des implémentations, de la compilation, de l'exécution, réalité des modèles de sécurité, etc.

Les résultats obtenus appellent à d'autres futurs efforts. Concernant l'axe autour du cryptosystème de McEliece, porter l'attaque sur le candidat pour le concours du NIST, améliorer les propositions de mesures pour une implémentation en boîte blanche, ou améliorer l'attaque telle quelle sont de futurs travaux potentiels. À propos de l'isolation des contenus sur Android, les contre-mesures proposées, suffisantes techniquement, restent surtout soumises à des arbitrages commerciaux et industriels, basés notamment sur la possibilité ou non de modifier le système, le désir de garder les solutions accessibles, ou de maintenir la compatibilité avec le service dédié à l'énergie. Continuer à enquêter sur les enjeux de l'énergie dans ce système d'exploitation et leurs liens avec la sécurité est cependant prometteur. De manière plus large, investiguer sur les compromissions dues à l'assemblage et la coopération de composants non testés ensemble est un travail d'ores et déjà en cours.

La distinction entre les contenus pouvant exploiter les mécanismes de sécurité matériels et les autres condamnés à viser des opérations en boîte blanche est lourde pour les fournisseurs de services. Les conséquences sur l'industrie et la compétition sont monumentales, puisqu'elles divisent inégalitairement le secteur en deux groupes : les acteurs ayant accès aux SEs et TEEs, et les autres. Ce n'est cependant pas une fatalité. L'organisation de l'amorçage sécurisé sur ordinateur a montré qu'il est possible de proposer une fonctionnalité de sécurité basé sur du matériel, sans ségréguer les acteurs. Les constructeurs de cartes mères signent les contenus et autorités tierces, et laissent avant tout le dernier mot aux utilisateurs.

L'assimilation de la multiplicité des contenus aux diverses origines tierces comme étant propice aux dangers est courante chez les détracteurs de l'ouverture des mécanismes de sécurité matériels. Mais ce n'est pas nécessairement le cas en pratique, une fois qu'une forte politique de sécurité et une conception du système rigoureuse ont été mises en place. Aujourd'hui, il existe de rares mais notables initiatives pour ouvrir ces accès. Cependant de manière générale, les pratiques anticoncurrentielles et privatrices restent la norme, comme l'attestent les affaires judiciaires se déclarant de plus en plus.

L'avènement des ordiphones a inauguré de nouvelles restrictions sur les utilisateurs vis-à-vis de leurs propres systèmes. Ils ont d'abord dû apprendre à faire des manipulations avancées et inhabituelles pour récupérer les droits administrateurs, ils doivent maintenant en faire le deuil définitif sur certains de leurs composants, régis par le secret et les accords de non-divulgation. Il n'est pas contesté ici que certaines opérations sensibles, par exemple liées aux paiements, ont besoin de logiciels vérifiés et validés. Il n'est pas non plus assumé qu'une part significative de la population soit en mesure de comprendre et prendre en charge par elle-même la sécurité de son ordiphone. Mais le marché actuel a établi une relation entre les constructeurs et la société publique basée sur le paternalisme, ou ces premiers imposent leurs visions et décisions à la seconde. Or, rétablir la liberté de choix auprès des utilisateurs est possible; ceux-ci pourraient choisir de déléguer les responsabilités de sécurité à un acteur de leur choix, que ce soit une entreprise, une association, ou une personne physique. Un tel chargé de sécurité pourrait sélectionner l'environnement logiciel des racines de confiance, désigner les contenus qui pourraient s'y installer, et concevoir les politiques de sécurité à l'intérieur et à l'extérieur d'elles. Une infrastructure à base de certificats, comme pour l'amorçage sécurisé au sein des ordinateurs, peut rendre concrète et techniquement possible une telle liberté de délégation. Les fournisseurs de service pourraient désigner les chargés de sécurité auxquels ils font confiance ou non. La transparence des mécanismes de sécurité pourrait être un argument clé auprès des utilisateurs et des tiers, aujourd'hui, il ne l'est pas.

Contents

Acronyms i

Remerciements v

Résumé vii

Abstract ix

Synthèse en français xi
- 0.1 Introduction xi
  - 0.1.1 Contexte et motivation xi
  - 0.1.2 État de l'art xiii
- 0.2 Contribution xv
  - 0.2.1 Offensive sur le cryptosystème de McEliece et propositions pour implémentation en boîte blanche xv
  - 0.2.2 Confidentialité des processus sur Android xvii
- 0.3 Conclusion xx
  - 0.3.1 Leçons xx

Foreword xxvii

I Introduction 1

1 Context and motivation 3
- 1.1 Embedded devices 4
  - 1.1.1 Traditional definition 4
  - 1.1.2 The shift of the definition 4
- 1.2 Sensitive features concentration 5
  - 1.2.1 Secure communication channels and storage 5
  - 1.2.2 Biometrics and identification 5
- 1.3 Hardware security features 6
  - 1.3.1 Roots of Trust 6
  - 1.3.2 Hardware Security Modules 7
  - 1.3.3 Trusted Platform Modules 7
  - 1.3.4 Trusted Execution Environments 8
  - 1.3.5 Secure Elements and secure enclaves 9
- 1.4 Electronic banking and Commercial Off-The-Shelf devices 10
  - 1.4.1 Traditional banking on the web 10
  - 1.4.2 Paying with a smartphone 11
  - 1.4.3 Receiving payments on a smartphone 12
  - 1.4.4 Regulation and security standards in the electronic banking industry . . . 12

1.5 Software-based security: the imposed path . . . 13
1.5.1 Developing secure solutions on smartphones as a third party . . . 13
1.5.2 The consequences of an unequal secure paradigm . . . 13
1.5.3 The redefinition of the industry's security models . . . 13

2 State of the Art **15**
2.1 White-box cryptography . . . 16
2.1.1 The historical black-box model . . . 16
2.1.2 The white-box model . . . 16
2.1.3 Precomputation . . . 16
2.1.4 Size circumvention . . . 17
2.1.5 Security measures . . . 18
2.1.6 The limits of encodings and linear transformations . . . 19
2.1.7 Expectations around white-box implementations . . . 19
2.1.8 The ephemeral security of white-box implementations against black-box equivalents . . . 20
2.1.9 Hardware attacks and their migration to the white-box model . . . 20
2.1.10 Reverse engineering and analysis circumvention through simulated hardware attacks . . . 21
2.1.11 Asymmetrical white-box implementations . . . 21
2.1.12 Exploitation in a defense in depth approach . . . 22
2.2 Confidentiality of processes in Android . . . 22
2.2.1 Abstraction choices . . . 22
2.2.2 Applications supply chain and moderation . . . 23
2.2.3 Threats distribution over the several, distinct abstraction layers . . . 23
2.2.4 Isolation breach in practice . . . 24
2.2.5 Code entry spying . . . 24

II Contribution **27**

3 Offensive and white-box propositions for the McEliece cryptosystem **29**
3.1 The McEliece cryptosystem . . . 30
3.1.1 Interests . . . 30
3.1.2 Definition . . . 30
3.1.3 Performances and footprint . . . 31
3.1.4 Legacy . . . 32
3.2 Offensive . . . 32
3.2.1 Cryptanalysis history . . . 32
3.2.2 Hardware attacks history . . . 33
3.2.3 Motivations for offensive research . . . 34
3.2.4 Preliminary inspections . . . 34
3.2.5 Determining hot spots . . . 34
3.2.6 The permutation matrix in the McEliece cryptosystem . . . 35
3.2.7 Exploiting the Arm instruction set . . . 37
3.2.8 Information extraction tactic . . . 38
3.2.9 Operands order explanation and determination . . . 42
3.2.10 Practical example when $accumulator = accumulator - matrix\ line$ . . . 43
3.2.11 Practical example when $accumulator = matrix\ line - accumulator$ . . . 47
3.2.12 Resulting metrics . . . 49
3.3 White-box measures . . . 50
3.3.1 Context and motivations . . . 50

3.3.2 Decryption 51
3.3.3 Encryption 53

4 Inter-process confidentiality in Android **55**
4.1 Power management in embedded devices 56
4.1.1 Power saving and its stakes 56
4.1.2 The challenges of managing rechargeable batteries 57
4.1.3 Better measurements for better decisions: the fuel gauges 59
4.2 Risk assessment 60
4.2.1 Fuel gauges hardware integration 60
4.2.2 Android's architecture and abstraction stack 62
4.2.3 Android's policy around fuel gauges 64
4.2.4 Identified risks 66
4.3 Sensitive data recovery through fuel gauges 67
4.3.1 Testing tools and experimental conditions 67
4.3.2 Data recovery 69
4.3.3 Data exploitation 70
4.3.4 Discussion about the results 73
4.3.5 Countermeasures 75

III Conclusion **77**

5 Outcomes **79**
5.1 Analysis 79
5.1.1 Recontextualization 79
5.1.2 Synthesis of the contributions 81
5.1.3 Impact of contributions 82
5.2 Prospects 82
5.2.1 Future works 82
5.2.2 Industrial and competition considerations 83
5.2.3 Societal considerations 84

Manuscript and defense reports **87**
Pascal Lafourcade's manuscript report 88
Clémentine Maurice's manuscript report 91
Thesis defense report 94

Foreword

With electronic devices being more and more widespread and personal, the stakes around computer security took new shapes and forms. This manuscript aims at restoring the state of researches on this particular angle, while also describing the contributions made on this topic. Since this matter is looked into both by the academic world and the industrial one, properly stating the interests and positions of each actors is essential. Hence why in this manuscript, technical considerations are frequently surrounded with situational ones.

This PhD thesis is split in three parts: Introduction, Contribution, and Conclusion. The Introduction is subdivided in two chapters. Chapter 1 is dedicated to restoring the historical context around computers and machines made for a wide audience. Hardware and mobility improvements helped making devices quite personal. This had an impact on how security is envisioned by all industrial actors. Thus, hardware equipment usually restricted to the military or professional worlds were progressively integrated in systems for the civil society. How it might be used for sensitive operations is described, with the help of the classic scenario of digital payments. However, a major counterpoint is emphasized: not all developers get the same possibilities and accesses regarding security features. This situation led to a growing interest towards security models such as the one referred to as a white-box, where all operations happen under public scrutiny. The second chapter of the Introduction, Chapter 2, is devoted to the state of art, both about the white-box security model and about the confidentiality of contents on Android off-the-shelf products. First, the most common way of designing sensitive software, which is with confidentiality guarantees from the platform, is recalled. Then, the white-box paradigm is explained. Its technical bases, advantages and limitations are all detailed. The offensive methods deployed against it and the consequences on the real robustness of such implementations in practice are covered, too. The latest developments on its subject are also given. On the confidentiality of processes on Android, first the architecture and abstraction choices of this operating system are summarized. The varied hot spots for different threats are then listed, located, and discussed. Specifically, on the topic of contents isolation, practical considerations are addressed, and illustrated with an emblematic case: the spying of secret codes entries.

The second part, dedicated to this thesis' contributions, is also split in two chapters. Chapter 3 is about attacks on the McEliece cryptosystem and how it could be relevant in a white-box scenario. The definition of this cryptographic algorithm, its inner working, along with the advantages it offers today are covered. The different improvements and offensives on its subject overtime are listed and analyzed. In particular, it is noted that attacks based on fault injections against its secret key were never achieved. After inspecting this particular topic, it is deducted that targeting implementations rather than the specification might give good results. In fact, subsequently, a successful offensive method focusing on a common way of implementing vector-matrix multiplications in $\mathbb{F}_2$ on Arm processors is introduced. Its multiple aspects are covered, and, for good and extensive clarity, two complete step-by-step examples based on different parameters are detailed, because the method to follow greatly differs depending on them. The consequences of such an attack are examined. In response to offensive behaviors like this one that might allow to circumvent simple obfuscation, propositions to make the McEliece cryptosystem more friendly to a white-box context are given. The impacts on both decryption

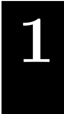
and encryption are explained. Chapter 4, about the investigations on content confidentiality on Android, starts a large presentation on the subject of power management in mobile devices, and why it can have substantial impacts on the security of a platform. A class of components, called fuel gauges, is introduced, and it is explained why constructors of smartphones and tablets more and more delegate responsibilities to them. A focus is provided on how these integrated circuits are implemented in Android products, both from the hardware and software point of view. Indeed, Android's abstraction stack, security policies, and dealing of applications-to-hardware accesses are explored and described. In consequence of it, three identified threats are announced. The last one, which is spying between applications, is then explored in details with a practical scenario: the spying of secret numerical code entries happening in other software contexts. After describing the experimental conditions, it is shown how it was achieved, in two steps: first, data acquisition, secondly, data exploitation. The obtained results are addressed and their consequences are discussed. Finally, varied countermeasures applicable in different abstraction layers are given, along with their corresponding advantages and disadvantages.

The last part, composed by Chapter 5, concludes this PhD thesis. Everything is first recontextualized to recent history. Then, the contributions made as part of this work are recalled and summarized, along with their impacts. The implications one could draw from them about the current state of computer security on personal off-the-shelf products are suggested. In a more prospective way, thoughts about the effects on the industrial world and on competition are expressed. Similarly, a discussion on societal aspects is explored, with some propositions. Finally, an approach on future works considering all this is detailed.

Part I

Introduction

Chapter



Context and motivation

Abstract

This chapter first exposes the traditional, original definition of embedded devices. Then the way it changed over time is detailed, along with the expectations around them. Indeed, a lot of them became more and more personal, and their different new usages are listed and explored here.

On the software side, they propose rich, multipurpose environments, able to run executable content from many sources, which can make developers fear the lack of control on such systems, from a security point of view. In consequence, the Root of Trust (RoT) principle is defined, as it is frequently cited as a solution for this kind of challenge. Then, the different hardware features than can possibly act like one are listed and for each of them, their historical context is detailed.

After this, the security stakes are recontextualized around the world of electronic payments. In a similar way, they used to happen only on industrial, professional machines, but this changed significantly in the recent years, as they started shifting to consumer grade platforms. The regulation framework and the different standards around them are recalled. Finally, a reflection is developed around the state of the market of secure solutions about payments, around the consequences of how it is beyond a commercial point of view, and around how security based on software only is becoming more and more considered.

Chapter content

1.1 Embedded devices 4
- 1.1.1 Traditional definition 4
- 1.1.2 The shift of the definition 4

1.2 Sensitive features concentration 5
- 1.2.1 Secure communication channels and storage 5
- 1.2.2 Biometrics and identification 5

1.3 Hardware security features 6
- 1.3.1 Roots of Trust 6
- 1.3.2 Hardware Security Modules 7
- 1.3.3 Trusted Platform Modules 7
- 1.3.4 Trusted Execution Environments 8
- 1.3.5 Secure Elements and secure enclaves 9

1.4 Electronic banking and Commercial Off-The-Shelf devices 10
- 1.4.1 Traditional banking on the web 10
- 1.4.2 Paying with a smartphone 11
- 1.4.3 Receiving payments on a smartphone 12
- 1.4.4 Regulation and security standards in the electronic banking industry . . 12

1.5 Software-based security: the imposed path 13
- 1.5.1 Developing secure solutions on smartphones as a third party 13
- 1.5.2 The consequences of an unequal secure paradigm 13
- 1.5.3 The redefinition of the industry's security models 13

1.1 Embedded devices

1.1.1 Traditional definition

Embedded devices, or embedded platforms, are prominent systems in various sectors, whether it is in the industrial, professional, or personal domain. They are defined as platforms dedicated to a specific task: sometimes controlling mechanical devices, sometimes processing data for instance. They can be integrated in a larger master system. They also can be powered by a reliable, fixed power source, or by an intermittent, mobile one. External factors along with their working constraints often require them to exploit Real-Time Computing (RTC) techniques in order to achieve their task. Examples of embedded platforms range from robotics in an industrial context, to electronic watches and street furniture.

As they are frequently dedicated to a single task, embedded devices' software often run on the hardware without abstraction. This is referred to as the bare-metal approach. This way of functionning is sensible when only one process needs to be managed. In this scenario, a Hardware Abstraction Layer (HAL) library might be exploited. Otherwise, Operating Systems (OSs) can be used, particularly when multiple, different processes and users need to share the platform's resources: processor, memory, and peripherals. But even when this is the case, it will likely be a small one in terms of size, with straightforward features, but strong guarantees, such as FreeRTOS [Fre], TinyOS [Tin], VxWorks [VxW], or ones specifically dedicated to security-oriented components [BGG21]. Even if an embedded platform does not have an OS by itself, it can collaborate with one. This is notably the case for hardware accelerators, like Graphics Processing Units (GPUs) for instance: complex and multipurpose environments control them.

As production costs for these devices decreased, and since they dramatically improve productivity, embedded devices were greatly developed and implemented in industries. Their typical use and environment in this context are highly controllable and predictable. They often reproduce one or a few simple tasks, and do not manipulate various data of different nature.

1.1.2 The shift of the definition

In the 1990s however, the popularisation of one specific type of embedded device has turned this field in different directions. This period saw the development of mobile phones, right after the one of portable audio players. These systems were considered embedded devices with special operational constraints from the start. The main one would be portability, and the consequences it has on size, weight, and the power supply. A portable phone indeed needs to be small, lightweight, and to run on a battery with limited charge. This type of platform also has to manage a lot of radio, wireless functions.

With the influence of mobile phones, the domain of embedded devices became less about platforms dedicated to one specific action. More and more features were proposed by them, which became expected from this type of machines. At the same time, for a lot of them, the main target switched from the industry and from professionals to regular, individual citizens. This difference brought a change in the featured functionalities and, most importantly, the processed data.

While most mobile phones before the 2000s were limited to a few features with little customization available, the ones designed after this period did not have these limitations. Content and executables from third parties, meaning not from the constructor, became widely available. Besides being heavily configurable by mobile operators, cellphones started accepting software from many developers and could access content from a large, varied choice of sources thanks to the Wireless Application Protocol (WAP) [Ope]. Also, the end users themselves could greatly personalize the executed tasks.

Smartphones were popularized in 2007 with the first iPhone from Apple, and in 2008 with the first one based on Android from Google and High Tech Computer (HTC) Corporation. This

step largely contributed to the shift in the definition of embedded devices, as smartphones are considered to be ones. They are nonetheless platforms made from the start to accomplish an extremely wide range of tasks. Their form factor and their portability turned them into very personal devices, with strong power efficiency.

1.2 Sensitive features concentration

Recently, the use cases brought to smartphones have become more and more demanding. Digital assets and possibilities started to accumulate on these devices, and many sensitive industries developed an interest to implement services on them.

1.2.1 Secure communication channels and storage

First, these systems began proposing many more communication channels than phone calls and text messages. Smartphones now embed several ways to talk, powered by different companies, such as Signal or WhatsApp. They can also be based on open protocols and rely on self-hosted solutions, like the Element application. In any of these cases, privacy became highly regarded, and the end-to-end encryption of communications is now fully expected by users. But these measures are under the responsibility of each of the channels providers, as communications on phones is no longer a monopoly of telecommunication companies that respect the same specific standards.

Secure storage of sensitive data in general is also an expected standard today. Smartphones often hold files that should only be readable by the owner, failing which they would induce massive privacy damages. Thus, strong storage encryption has been normalized in these platforms, most of time implemented at the partition level, as default. Attack scenarios on phones henceforth consider this reality, and this class of devices thus became more trusted by users to hold sensitive data.

1.2.2 Biometrics and identification

Biometry-based authentication is now common in cellphones too. Its main use case is to unlock the phone more quickly and with less efforts than with Personal Identification Number (PIN) codes or passwords. The firstly used physical feature is the fingerprint. Later came facial recognition exploiting means more advanced than the simple front camera. In any way, the capture, storage and in general, managing of personal biometric data is very sensitive but became nonetheless common on these embedded devices.

Smartphones are now also considered useful for identification with state-level trust. Concretely, they are considered as equivalents of the state-issued physical identification cards. The France Identité [Fra] application and the European digital identity wallet project [Eur23] for example, have the ambition to be exploitable in any official situation where a citizen must be identified. A hijacking of this kind of mechanism can thus have substantial consequences. At the same time, the Système Électronique de Saisie de l'Assurance Maladie (SESAM-Vitale) economic interest group is currently testing its Carte Vitale [Car] application in some of France's territories. Its goal is to dematerialize the Vitale card, which allows any eligible french citizen to authenticate himself and benefit from a direct settlements when getting health care.

Identification for the personal uses or for private services also rely increasingly on smartphones. Multi-Factor Authentication (MFA) can be based on text messages sent to them, or on Time-based One-Time Passwords (TOTPs) stored on them, for instance. WebAuthn, on the other end, is a relatively new method for primary authentication on the web, that depends on the conservation and exploitation of keys by such personal devices. While TOTPs and WebAuthn are both publicly and openly enacted (the first one by a Request For Comments (RFC) from

the Internet Engineering Task Force (IETF) [MRPM11], the second one by a standard from the World Wide Web Consortium (W3C) [Wor]), some companies implement proprietary solutions, as it is the case with Microsoft Authenticator [Mic].

These are only the most prominent symptoms of the concentration of sensitive services and data on the people's smartphones. With this accumulation come major risks in case of cyber attacks, and the legislative environment is only built up along the way, either slightly early, or largely late. In response, mobile phones constructors needed to implement the requirements to offer all these possibilities.

1.3 Hardware security features

In general, when exploiting sensitive processes on information systems, hardware security features are considered obligatory [CP16]. Rich Execution Environments (REEs), based on multipurpose and applicative processors with the OSs they run, provide guarantees deemed insufficient. They have a wide attack surface, especially in software, and are more prone to hardware flaws, like fault injection, or side-channel exploitation. On the other ends, while hardware security features have long been restricted to industrial or military uses, they became more or less accessible to the general public, as personal devices popularized.

1.3.1 Roots of Trust

When considering a system running in an industrial environment, one can develop its security policy on the assumption that attackers will not be able to get physical and software access to it. Indeed, they are sometime not connected to any network at all, and located in facilities with stringent access controls. In this scenario, the architecture of the system is likely quite straightforward, relying on traditional, ready-to-use features such as memory protection and process isolation. In general these are enough to counter issues caused by bugs threatening confidentiality, integrity or availability.

However when considering devices put in the hands of the general public, the attack scenario becomes radically different. Every interface or access point becomes potentially subject of misuses. Even areas or operations usually trusted can be tampered by attackers: memories are a good target for hardware attacks, and executions often create side-channels.

Multi-user OSs have an administrator account, named root in the Unix-based ones. This account has all rights and permissions on the system, contrary to unprivileged users. This paradigm has been developed by focusing on the professional management of computers: an administrator is responsible for the computers of collaborators who cannot do whatever they want on them. However, the attack surface of a full-blown OS is most of the time considered so large that few actors actually decide to rely only on the administrative account to protect assets and operations. In the case of smartphones, the root account is usually locked by constructors, and buyers initially get less possibilities and freedoms in exchange for more security. A way to unlock phones is nevertheless almost always available, and in this case, the final user becomes *de facto* able to do everything.

What can a system trust in this context? The answer initially is nothing. However, the features mentioned below (Hardware Security Modules (HSMs), Trusted Platform Modules (TPMs), Trusted Execution Environments (TEEs), and Secure Elements (SEs)) all try to represent an anchor point one can count on. This concept is called the Root of Trust (RoT). It refers to components, features or environments that can be trusted, as small as they are, in a rather hostile context. Using cryptographic methods, their safeness can actually be extended to the rest of the platform. The most emblematic example is the one of integrity verification and secure boot: before being executed, the checksum of every firmware, driver, bootloader, and kernel is calculated and then verified by the RoT. Having your software environment attested

by it allows for more confidence in it. This propagation of trust to the rest of the platform is why these features are designated as roots.

RoTs fit in a particular context. Since the beginning of Information Technology (IT), end users had to be controlled in order for a device to be considered secure. Hence the fact that the people's Personal Computers (PCs) is considered dangerous or not suitable for many applications [Tar23]. The same fate was met by smartphones, at least the ones providing freedoms and control to users, such as the ones based on Android. However as the industrial urge to exploit these personal mobile computers became stronger and stronger, RoTs appeared as a protected anchor point for commercial exploitation; a shelter against the users themselves.

One particular domain which is emblematic of this phenomenon is banking and electronic payments. Its advent in the 1980s relied a lot on devoted machines, like dedicated smart cards and dedicated terminals. However, just as the other use cases such as identification, its dematerialization depends a lot on personal systems. Lots of attention has been paid to smartphones and their environment, described as Commercial Off-The-Shelf (COTS) devices, since they can be found on the shelves of any electronic store, are quite affordable considering a median income, and can be bought easily without constraint.

1.3.2 Hardware Security Modules

Hardware Security Modules (HSMs) refer to complete systems dedicated to secure operations, mainly cryptography. They usually have the form-factor of a full, rackable server, or the one of a whole Printed Circuit Board (PCB), for example compliant with the Peripheral Component Interconnect Express (PCI Express) standard. They are able to protect cryptographic keys and operations, and are expected to provide resistance against tampering attempts. Execution of third-party code is rare; if present, stringent measures have to ensure its authentication and integrity.

HSMs were introduced at the end of the 1970s, along with the Data Encryption Standard (DES) [Nat99]: efficient and protected implementations of it were needed. However it was mostly produced for the military sector and for uses at the governments level. Commercial HSMs really appeared in the 1990s, in order to secure financial and banking systems such as Automated Teller Machines (ATMs) for instance. There were also urgently needed to secure communications on the newly-developed World Wide Web, and were thus vastly deployed in servers and at certificate authorities. The payment terminals frequently encountered in stores since this period can also be considered as HSMs. Overall, HSMs are still common today in the servers rooms of sensitive industries.

1.3.3 Trusted Platform Modules

Trusted Platform Modules (TPMs) are embedded processors dedicated to operations related to cryptography. To be precise, TPM is the name of a standard [ISO15] describing their functionalities and interfaces. Such a component needs to include features such as asymmetric and symmetric cryptography, a Random Number Generator (RNG), time-keeping and hashing. While being able to process these kind of operations, TPMs are still considered passive devices, as they only reply to commands and do not control anything on their host. They cannot run arbitrary code; the set of features that they provide is limited and must be available as described in the standard. They must be isolated from application execution, possibly from the general processor itself. With time, they tend to get closer to it, from having a dedicated chip, to being integrated in the platform's chipset and more.

The premises of TPMs are the 1990s and the realization that many commercial applications running on PCs were going to require trusted functionalities. The Trusted Computing Group (TCG) released in 2003 the first version of the standard that was widely considered: TPM 1.1b. It was considerably deployed by 2005. While software protections were specified, measures

against hardware tampering were left to manufacturers. In order to keep costs as low as possible, anything not strictly required in the TPM was delegated to the application processor. The 1.2 version was developed and deployed starting 2005 in order to fix incompatibilities that could still lie in the software and hardware interfaces. Amongst others, protection against brute-force attacks was also added. The TCG began the development of version 2.0 at a crucial point: when SHA-1 (Secure Hash Algorithm) started to show weaknesses in the middle of the 2000s. Instead of enforcing a new specific alternative that would itself be subject to the same flaws in the future, the engineers preferred to prescribe agility regarding algorithms, that is, allowing algorithms changes without changing the standard. This design implied to make symmetric algorithms mandatory, as they were not at the time, and as the new unpredictability in data size made it no longer possible to use asymmetric ciphers only [ACG15]. TPM version 2.0 has represented a radical change comparing to the previous ones and was finally released in 2019. It has been designed to avoid the need for any major update in the future.

TPMs are mostly implemented in PCs, where they are useful for data and disk encryption along with integrity attestation. Microsoft has played an important role in the acceleration of their deployment. In 2021, they announced Windows 11 would require the presence of a TPM conforming to version 2.0 in order to be installed and to run, after imposing to integrators since 2016 to implement one in PCs sold with Windows preinstalled. Cars, which have information systems that get more and more complex with time, are also an important target for these components, since they are intensely exposed to attackers and the threats they are facing can have significant and direct consequences.

The TPM standard can also be implemented by using hypervisor technology. Indeed, a fundamental aspect of these is that a Virtual Machine (VM) should be not be able to influence its host. This axiom can be exploited to isolate a Virtual Machine from its virtual TPM. This approach is considered less secure as using a more conventional TPM implementation, but it is highly scalable as it relies less on hardware. Hence the fact it is a solution mostly considered in cloud products.

1.3.4 Trusted Execution Environments

Trusted Execution Environments (TEEs) are security oriented environments in which programs benefit from many security guarantees. Instead on relying on hardware components distinct from the main processor, they are based on specific areas or modes of the primary system. Executables exploiting these mechanisms thus still run on the same chip as the others, but cannot be inspected or controlled by the general application OS, and the protection of their context is covered by hardware features. When benefitting from a TEE, programs get isolated execution, integrity verification, and confidentiality on them and their assets. Such environments can also be configured so that specific hardware resources voluntarily become unreachable.

This paradigm was first unilaterally introduced by Arm in 2004 with its TrustZone technology [AF04]. It was then formalized by the Open Mobile Terminal Platform (OMTP) in 2009 [Ope09]. Its standardization was initiated by the GlobalPlatform authority alone in 2010 [Glo10], and continued in collaboration with the TCG in 2012 and later. In order to support TEEs, Intel introduced in 2015 the Software Guard Extensions (SGX): an additional set of instructions on top of the classic x86 one. However, starting 2022, it was progressively discontinued on consumer products. The discovery of a consequent number of attacks against its design [BKS+22, WKPK16, ERAP18, SMA+20, MBH+20, RMR+21, BMW+18, SVSPF20, CS13, SWG+19, SYG+19, LKO+21, MOG+20, SAS+19, CCX+19, CYS+21, BWK+17, BCD+18, KFG+19, CVM+21, SLM+19] lead some to infer it might be because of issues with its security guarantees [New]. In 2021, Intel began proposing the Trust Domain Extensions (TDX) [Int], another set achieving similar goals, but with different methods.

Despite being both considered as technologies providing TEEs, SGX and TrustZone are

conceptually different. An environment provided by TrustZone is quite large, and requires its own OS dedicated to it, next to the general applicative one. This OS then manages its applications and their operations on the same principles as a classic one, but in this special, secure environment. Examples of such OSs include Trusty OS from Google [Goob], QSEE from Qualcomm [KM22], Kinibi from Trustonic [Tru23], or iTrustee from Huawei [ANS]. On the other end, with SGX, these are not needed: the main application OS is allocating processing time and memory to the trusted environments just like with any other application. It however cannot know how these resources are used and what is being done with them.

TEEs were the first security feature to be widely deployed in products for the general public. This was what helped to define their most popular use cases. SGX were exploited a lot for Digital Rights Management (DRM), notably for Blu-Ray discs. On mobile phones, fingerprint recognition and management guidelines were given by Google starting with the sixth version of Android, and the use of TEEs for it was decreed as mandatory from the very beginning [Goo15]. As of today, TEEs are still one of the authorized security features for handling any biometric sensor and its data [Goo23].

Unlike TPMs, TEEs are integrated in the very core of the general application processor. Despite this, various processors manufacturers such as Intel, AMD and Qualcomm have implemented the TPM standard in a software version embedded in their own TEE technology. This type of TPMs, called firmware Trusted Platform Module (fTPM), allows to get TPMs features not based on hardware requirements (that is, the chip itself). While this accelerated dramatically the deployment of TPMs, it is also considered less safe as a usual, hardware implementation of the standard, and can interfere with the performances of general applications. However, Microsoft still decided to accept fTPM implementations in their requirement for Windows 11 installation and use.

When a TEE is available on a platform, every content tends to become considered as either secure or not secure, without in-between. Also, as more and more processes target these secure environments, the larger becomes the Trusted Computing Base (TCB). This notion describes the set of critical and sensitive assets of any nature that is present on a platform, and that could compromises it in its entirety if it gets hijacked. To counter this, Google unveiled in 2022 the Android Virtualization Framework (AVF) project. It consists in exploiting a hypervisor that is not running in the TEE but has a higher privilege than the OS. Sensitive applications can thus be executed in a separated VM. Bugs or undesirable behaviors in Android would not affect both the hypervisor or the protected applications anymore. On one hand, the AVF relies essentially on the Exception Levels (ELs) of Arm processors and make them exploitable behind a common Application Programming Interface (API). On the other, today it cannot replace the usual TEEs as some devices require them to be contained by Systems-on-Chip (SoCs).

1.3.5 Secure Elements and secure enclaves

Secure Elements (SEs) are platforms dedicated to secure operations and storage. Like TPMs, they are supposed to represent an independent, distinct architecture on their own. However, unlike them, their set of features is not restricted by a standard. They can thus host third-party applications, under stringent conditions of authentication, certification, and integrity verification. Conceptually, they can be thought of as very small HSMs, small enough so that they can be integrated in smartphones or even directly in the same package as a component dedicated to something else. As expected, they are required to have tampering protections, including against hardware attacks.

Historically, SEs are widely deployed to the general public since the 1990s, as chip cards can be considered as ones and were largely delivered since this period. The same can be said with Subscriber Identity Module (SIM) cards, which are inserted in cellphones since this same decade. The strong links between smart cards and SEs explain why the GlobalPlatform organization,

which standardizes a lot of the former's aspects, got *de facto* the same authority over the latter. It also indicates why on smartphones, embedded Secure Elements are often integrated close to the Near Field Communication (NFC) modem: it is one of the two communication channels available on smart cards.

Secure enclave is a term introduced by Apple in order to describe a concept similar to SEs [GSY14]. It has the same properties, but is directly included in their SoCs (while still being separated from the general processor) [MSW16]. Also, their software characteristics along with their interfaces are not subjected to GlobalPlatform's specifications and are only up to Apple's proprietary decisions.

For long, SIM cards were the only SEs that could be found in phones. It lead to tensions in the industry, as these cards are completely controlled by network operators: other companies would become completely dependent on them and their permission to exploit them. But then embedded Secure Elements (eSEs) appeared directly in smartphones. The first Android phone to integrate one was the Nexus S in 2010, but it took several years after this for them to become common in released models. Indeed, the Host Card Emulation (HCE) technology, introduced in Android in 2013 by Google, temporarely relieved industrials' needs to gain accesses to SIM cards, by allowing NFC communications from applications, bypassing them. However sensitive processes like electronic payments and state-level identifications imposed the requirement of SEs because of the security downgrades implied by HCE. SIMs cards, on their end, progressively stopped being removable, as the Embedded Subscriber Identity Modules (eSIMs) specification turned them into chips soldered on the phones' PCB, starting 2016. From 2020, the eSIMs began to be merged with the components already present in smartphones, like secure enclaves or SEs, resulting in a new designation: the Integrated Subscriber Identity Module (iSIM). This allows to save more space on PCBs.

On Apple's side, the iPhone 5s, in 2013, was the first one to contain a secure enclave. It was implemented to secure its fingerprint-based authentication feature. On their following device, the iPhone 6 in 2014, in addition to the secure enclave, they added a conventional SE, compliant with GlobalPlatform's specifications, outside of the SoC but integrated with the NFC dedicated hardware. It is responsible for the processing and storage required to make payments. The secure enclave and the SE can communicate, notably when biometric verification of the user is necessary in order to authorize a payment [App22].

Despite being technically able to get and run third-party executables, SEs rarely do so in practice. Such collaborations require lots of negotiations along with commercial efforts, and in case of success, imply a delicate development with heavy audits and verifications. However a lot of use cases are quite generic. In 2021, Google unveiled the Android Ready SE Alliance, along with several SEs manufacturers. The point of this organization is to agree on a common set of open-source applets (meaning applications for such components) that would be present on the SEs integrated in Android smartphones, and available to use by applications on the phone. For example, the StrongBox applet thus enabled the secure, tamper-resistant storage of secrets in SEs to any third-party developer without the need to get into heavy industrial talks between corporation.

1.4 Electronic banking and Commercial Off-The-Shelf devices

1.4.1 Traditional banking on the web

It can be said that Commercial Off-The-Shelf (COTS) devices have been dealing with banking data since they existed. Firstly, it was simply the case because one could buy an item on the web, using his device, by manually entering his payment information. The main safeguard measure here is the use of encryption to protect the data during transmission. The most used protocol to achieve this is Transport Layer Security (TLS), with its predecessor Secure Sockets Layer (SSL),

just as on other types of machines, like computers for instance. While the sensitive information were also protected server-side with various industrial grade measures, the same cannot be said about the user-side. Keyloggers, spywares, or in general malicious software can be found in the end user's environment, and phishing attempts are widespread. These threats directly derive from the manipulation of cleartext banking data. This use case rely mostly on simple context isolation: on Android for instance, each application is used under a unique, specific Unix user associated to it. Thus, the classic software protection mechanisms found in Linux should prevent other applications to spy on the data manipulation ongoing in the web browser. These risks are also addressed by non-technical measures, like insurance mechanisms: the more safe is a system, the more cheaper it gets to be insured, and vice versa.

1.4.2 Paying with a smartphone

Today one can embed his smart card in his personal device. Such services were introduced by major smartphone industrialists in 2014 and 2015, allowing users to pay on classic payment terminals using only their phone, without their physical card. This feature is not based on an emulation of the specific chip card by the smartphone. Instead, at initialization, the former is tokenized, and the result is stored in the latter. A tokenization is an operation where a sensitive element is replaced by another data referencing to it, often generated randomly, that is a less hazardous asset since it, in itself, does not provide any information on what it points to. Thus, during a payment, credit card data are not disclosed to the terminal: only a token is exchanged, and the actual required references are recovered by the back-end payment infrastructure, which is completely under control of the industry. Even if tokens can seem less hazardous than actual credit card data, their manipulation is nonetheless highly regulated, which is why hardware security features are used in order to do so. While this is the most common technique used in the West, other ways of paying with a personal phone exist [LWP20].

Paying with a smartphone is a feature provided by several, distinct, private actors under different names: Apple Pay [App] or Samsung Pay [Sam] for instance. It is no coincidence that they refer to smartphones constructors, since they are the ones who have control on what can be done with their secure hardware. Indeed, the first one is based on eSEs [App22], and the second one is based on TEEs [Sam16]. They can thus integrate their own payment solution exclusively, and competition in the supply of this service is restricted due to the forbidden access of these secure technologies. Unless wireless network operators would open it by allowing the embedding of payment software in their SIM cards, which is not the case. The rare solutions actually doing this in production are for transportation. For example, during the end of 2010s, having a SIM card by Orange allowed to pay for Île-de-France Mobilités (IDFM)'s public transportation using a smartphone. Similarly, EZ-Link SIM cards still allow to do the same in Singapore. Google Pay [Gooa] (formerly Android Pay, formerly Google Wallet) stands out as a different approach: all banking data is stored on Google's servers and no sensitive logic happens on the phone. To make transactions, perishable tokens are generated online, downloaded, and can be relayed on the COTS device using just HCE, since they are less sensitive than actual banking data. This method is often said to rely on a "cloud SE"; however it implies a great dependency towards Google's servers, and still need some secure hardware in order to authenticate the user, whether it is though a password, biometrics or any other method.

While this is not common, some constructors of smartphones, like Xiaomi, sometimes allow the user to choose the technology payments will be based on. Depending on circumstances, choices can include HCE, SIM card use, or eSE use. Beyond the selection of which component will be responsible, the way transactions will be handled will necessarily change depending on it, since they do not offer the same guarantees.

1.4.3 Receiving payments on a smartphone

Meanwhile, using a smartphone as a payment terminal became a feature very much looked into by the industry, furthering the vision of these devices as secure enough to manage these responsibilities. Indeed, traditional Point Of Sale (POS) devices are designed with security for money transactions in mind from the start. They thus have robust components and software tailored for this use case, and their general design is done by considering a harsh attack scenario. Moreover, companies developing them have full control on their conception, from hardware assembly to the abstracted software layers.

Such control is not conceivable with smartphones. The actors designing the hardware are rarely the same as the ones developing applications. In that respect, even though NFC hardware appeared in personal cellphones in the beginnings of the 2010s, the reception of payments by emitted smart cards was not actively considered until many years later. Apple is currently deploying this feature on iPhones in France.

1.4.4 Regulation and security standards in the electronic banking industry

The payments industry security is heavily based on the self-regulation exhibited by this sector. Its actors follow requirements enacted by Europay Mastercard Visa & Co (EMVco) since 1995 and by the Payment Card Industry Security Standards Council (PCI SSC), since 2006. For instance, the latter produced the PIN Transaction Security (PTS) program, that a company producing a POS terminal must strictly follow in order for it to be able to connect to payment networks. The history and evolution of this industry is dramatically linked to these two organizations and to the directions they point out. The development of payment acquiring on smartphone is not an exception, as shown by the security standards published by the PCI SSC on the subject:

- in 2018, the Software-based PIN entry on COTS (SPoC) standard was released. It describes how a COTS device can take in charge the entry of a PIN code all by itself, without any additional keypad. It is important to note that in the scenario considered here, the payment card is not read by the COTS product directly, but by a physical add-on device linked to it. Even if smartphones alone are not sufficient in this mode of operation, this document represented a considerable, first step towards payment acquisition on personal devices: a recognition that it could be considered.

- the Contactless Payments on COTS (CPoC) standard was released in 2019. It sets out the constraints to overcome so that contactless payments can be handled by COTS devices, without any additional equipment. However, it only considers payments without PIN code entry. Thus, in the vast majority of countries, it only authorizes transactions under a set, fixed amount, since this is the main Cardholder Verification Method (CVM) used in order to limit spending without authentication by the consumer. The description of a payment method where smartphones are sufficient by themselves really confirmed the focus of the industry on them.

- Lastly, in 2022, the Mobile Payments on COTS (MPoC) standard was released. It fully embraces the vision of smartphones as payment terminals. It combines SPoC and CPoC and adds many other considerations. A considerable one is contactless payments with PIN entry as the CVM without any add-on hardware, whether it is for card reading or for the PIN code entry. Offline transactions are also taken into account. Unlike the PCI SSC's previous documents, this one gives many liberties to designers of the secure systems. It has all the aspects of the ultimate standard on the subject, championing smartphones as future-proof platforms for electronic transactions.

1.5 Software-based security: the imposed path

1.5.1 Developing secure solutions on smartphones as a third party

Smartphones are establishing themselves as essential elements of the future of payments. They also currently represent a harsh battlefield that all industrial actors do not face on equal footing. This is due to the context around hardware security features:

- HSMs and TPMs are not implemented in phones.
- TEEs represent efficient solutions for payments and secure booting. However they are not accessible to third parties, as they cannot develop an executable and deploy it on them. These environments are exclusively under the control of SoCs constructors, and sometimes smartphones constructors.
- SEs can also ensure safe payments and secure booting. But third parties are also prevented from accessing them. This is true for SIM cards, which are controlled by mobile operators, but also for eSEs, which are exclusively exploited by constructors of such components and of smartphones. While the Android Ready SE Alliance initiative might help with the most common use cases, it will not allow to implement arbitrary code in them. Reaching the insides of secure enclaves is obviously even less conceivable.

When developing sensitive software, all this makes any company that is not a constructor of smartphones or components doomed to operate on untrusted REEs. The inequalities in the reachable features implies a major bias between actors in their capacity to provide safe contexts for their algorithms.

1.5.2 The consequences of an unequal secure paradigm

The consequences of such differences are tough for the payment industry, and they also have geopolitical implications. As of today, France's payments infrastructures are unusually self-reliant. It is one of the few countries to have its own national payment scheme, Cartes Bancaires (CB), which provides 65% of all transactions made by citizen for everyday consumption [GIE]. Most of the Points Of Interaction (POIs) found in its territories are designed and provided by the French company Ingenico, and the leading manufacturer of chip cards, Thales Digital Identity and Security (formerly Gemalto), is from the same country. Many geopolitical conflicts showed how international payments systems and services can be instrumentalized in this kind of context [RD23]. Relying on solutions and infrastructures not even reachable by a country's industry thus become a clear and direct threat for sovereignty. Payments on COTS devices is at risk of only being affordable to a highly closed club of actors.

1.5.3 The redefinition of the industry's security models

It is then logical to observe a particular emphasis by the industry on solutions not relying on hardware mechanisms. Here, too, the publication of security standards gives an accurate picture of the major topics in the world of payments. In 2018, EMVco released the Software-Based Mobile Payment (SBMP) standard. It describes security certification processes intended at solutions and components independent of closed features from the environment. For instance, the Software Protection Tool (SPT) category is intended at components dedicated to protect products using only software methods. Getting such a certification on a software module is paramount for industrial actors, as it is recognized by the PCI SSC and allows to aim for a SPoC, CPoC or MPoC certification for a wider product using it. Symbolic of the challenges faced by the payment world, this track of development is quite popular, as the 100$^{\text{th}}$ evaluation certificate for SBMP was issued in November 2022 by EMVco.

Cryptography is the prime example of a sensitive process needing to be protected. While the scientific community around it is devoted a lot to the threats posed by quantum computers, the management of such algorithms in environments as complicated as smartphones is nonetheless an important matter, and these two subjects are not exclusive. At the beginnings of the 2000s, a new paradigm emerged in this domain: white-box cryptography. In contract to the black-box model, where a cryptographic component is considered as an impenetrable system on which an attacker cannot get any information besides what is going in or out, white-box cryptography considers attackers as able to read, write and manipulate anything inside or related to the algorithms. Despite its considerable concessions on robustness and durability, the whole context around security in COTS devices made this model very much looked into by the industry, as it frees cryptographic algorithms from any hardware consideration. Is secure computation on REEs possible?

Synthesis

In this chapter, the historical definition of embedded devices is recalled, and it is developed how it progressively changed toward systems with constraints like the ones around mobility, especially since smartphones. As these became drastically common in society, all the sensitive features they now accumulate are detailed.

Services providers often look for RoTs in these machines. This principle is explained, and all the hardware security features able to provide these are listed and examined. In order to provide a concrete example of this situation, the use case of payments from and to COTS products is explored, from a historical, technical, and even regulatory point of view. Finally, the stakes and constraints around sensitive services by third parties on these devices are reviewed and detailed.

Chapter

State of the Art

Abstract

This chapter first describes the historical context in which the white-box security model appeared. It explains its principles, and its technical foundations, heavily based on precomputation. Its limit are also detailed, especially around the encodings and linear transformations, often used in them. The expectations around white-box implementations are discussed as well, as they often are less clear than one could guess. Their overall security is examined, notably in comparison to models based on black-boxes. In particular, it is recalled how hardware attacks can be ported to the white-box paradigm, and how security flaws on an implementation can allow attackers to circumvent obfuscation and protection technologies, making it unsustainable. A glimpse on the developments of asymmetrical cryptographic white-boxes is also given, and the defense in depth approach, which is a popular in this kind of context, is addressed.

Then, the subject of inter-processes isolation is explored, with a focus on Android. The general security architecture of Android is explained, as well as the many stakes around the distribution of executable contents in this platform. State-of-the-art breaches in such devices are described. A focus is finally given on the spying of a code entry happening on another process as a malicious one.

Chapter content

2.1 White-box cryptography 16

2.1.1 The historical black-box model 16

2.1.2 The white-box model 16

2.1.3 Precomputation 16

2.1.4 Size circumvention 17

2.1.5 Security measures 18

2.1.6 The limits of encodings and linear transformations 19

2.1.7 Expectations around white-box implementations 19

2.1.8 The ephemeral security of white-box implementations against black-box equivalents 20

2.1.9 Hardware attacks and their migration to the white-box model 20

2.1.10 Reverse engineering and analysis circumvention through simulated hardware attacks 21

2.1.11 Asymmetrical white-box implementations 21

2.1.12 Exploitation in a defense in depth approach 22

2.2 Confidentiality of processes in Android 22

2.2.1 Abstraction choices 22

2.2.2 Applications supply chain and moderation 23

2.2.3 Threats distribution over the several, distinct abstraction layers 23

2.2.4 Isolation breach in practice 24

2.2.5 Code entry spying 24

2.1 White-box cryptography

2.1.1 The historical black-box model

Usually, a fully closed and controlled environment is considered mandatory to allow for the execution of secure operations. This is because this way, sensitive contents and data are not exposed to unwelcome eyes. Unexpected behaviors or mechanisms would not help in leaking any information. This view of a system is referred to as the black-box model, as an attacker should not be able to understand or grasp any information about its insides. Since the system is necessarily delimited, the content entering it or exiting it is not subject to such conditions, and is considered easily viewable. TPMs and SEs are typical example of this concept: the software implementations they are running are rarely publicly documented. Even if they were, hardware protections are put in place to prevent scrutiny or tampering with their insides, so that the security of these components do not rely solely on obscurity. However the way they are designed completely acknowledges that their communications with other actors can be freely read. This is for instance the case with modules complying with the ISO/IEC 7816 standard [ISO11].

2.1.2 The white-box model

In 2002, Chow *et al.* introduced the white-box model, first by releasing an Advanced Encryption Standard (AES) implementation [CEJP03b], secondly by releasing a DES implementation [CEJP03a]. In this paradigm, having data and execution protected is no longer considered: it is admitted that a malicious actor will be able to read or modify any data or control flow. More clearly, it implies that attackers can read or write to any memory, at any moment, change the control flow precisely, and instrumentalize the program in question. In the case of a cryptographic implementation, for example, the goal thus becomes to operate in plain sight, without compromising secrets such as the private key. The algorithms steps are already known by everybody since the specifications used are public. White-box implementations aim at being compliant with them despite these constraints: the same inputs should result in the same outputs. A valued advantage they have is that they do not rely on hardware security mechanisms, and are consequently easy to deploy on a large, varied fleet of products with different designs.

2.1.3 Precomputation

Surprisingly, such open cryptography was initially approached using quite simple techniques. Targeted algorithms are essentially converted to sequences of precalculated steps, and the tables containing these precalculated results are encoded in an ingenious way so that these originally useless operations cancel each other, allowing for the compliance with the original specification. Nonetheless, lots of tactics must be deployed to make this achievable. For instance, the AES works on data blocks with a size of 128 bits, and the DES works on data blocks with a size of 64 bits. Precomputing operations on these whole blocks requires considering 2^{128} and 2^{64} entries respectively, which would not be possible for two reasons:

- the time required for it is not reasonable. During the creation of the algorithm, a processor would need to compute these numbers of results for each considered step. If 2^{64} is reachable under a large amount of time and under specific conditions [Tez22], it is not in a production and industrial context, while 2^{128} is not, at all.

- the generated data size, which corresponds to $number\ of\ entries \times size\ of\ entries$, is completely inaccessible for any storage media, both for 2^{64} and 2^{128}.

2.1.4 Size circumvention

It is thus required to reduce the number of entries in each precalculation table. To do this, instead of considering the whole data block, one should consider a smaller unit of data. This is more or less complex depending on each operations. `SubBytes` and `AddRoundKey` for the AES, along with `Expansion`, `Key Mixing` and `Substitution` for the DES are operations where it is simple: the considered unit is already smaller than the data block, and the precomputation can be done in a straightforward way without issues. For example, in the `SubBytes` operation, the substitution is happening byte-wise, and it is possible to generate tables containing $2^8 = 256$ entries.

However, precalculation is more complex on operations like `MixColumns` in the AES. This step can be assimilated as a matrix multiplication on blocks of 32 bits, or on four 8-bits elements. Thus, if one represents a quarter of the input block as x_0, x_1, x_2 and x_3, and the outputs as y_0, y_1, y_2 and y_3, then it can be written as:

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

This operation can be split down to calculations based on 8 bits, with the assembling of the results using XORs. Indeed, the output is the addition of four terms, such as:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = x_0 \times \begin{bmatrix} 2 \\ 1 \\ 1 \\ 3 \end{bmatrix} + x_1 \times \begin{bmatrix} 3 \\ 2 \\ 1 \\ 1 \end{bmatrix} + x_2 \times \begin{bmatrix} 1 \\ 3 \\ 2 \\ 1 \end{bmatrix} + x_3 \times \begin{bmatrix} 1 \\ 1 \\ 3 \\ 2 \end{bmatrix}$$

Each of these terms is a function with inputs of 8 bits and outputs of 32 bits. Their size is consequently affordable, as each of them only need to store 256 possibilities. The additions are XOR operations precalculated for two 4-bits inputs, implying a size of 256 elements here too, with 4-bits outputs. By deconstructing the whole matrix multiplication in smaller steps, one can avoid the exponential behavior of the tables' sizes, only to keep multiple, smaller ones:

- 144 tables ($4 \times 4 \times 9$) dedicated to multiplication:
  - as explained earlier, 4 multiplications with an 8-bits input are required to get a 32-bits matrix multiplication
  - there are 4 32-bits elements in the 128-bits state of the AES
  - the `MixColumns` operation is applied 9 times in the AES
- 864 tables ($8 \times 3 \times 4 \times 9$) dedicated to additions:
  - since XOR tables take two 4-bits input, 8 of them are needed to process two 32-bits elements
  - as explained earlier, 3 32-bits XOR operations are required to reproduce a 32-bits matrix multiplication
  - there are 4 32-bits elements in the 128-bits state of the AES
  - the `MixColumns` operation is applied 9 times in the AES

The process of making a 32-bits matrix multiplication affordable to precomputing is illustrated in Figure 2.1. It is also explained in details in [Mui13], among others.

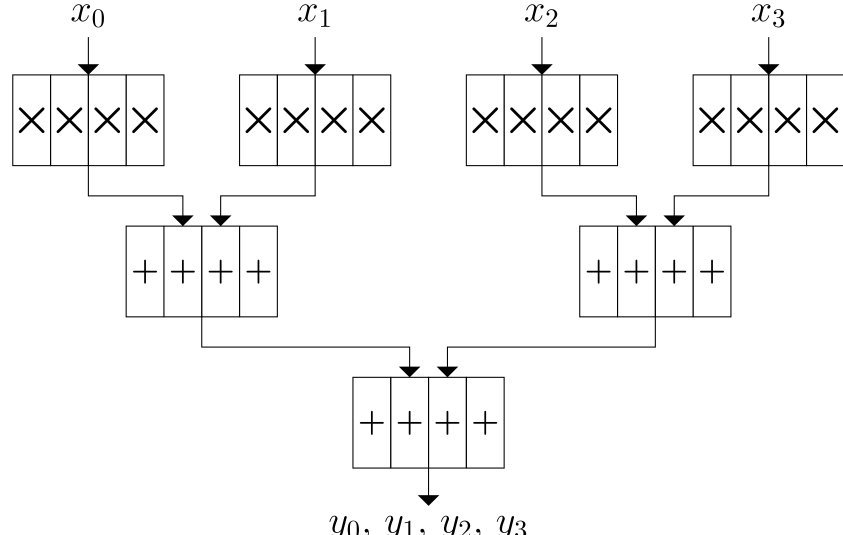


Figure 2.1: Depiction of the tables organization required in order to precompute a 32-bits matrix multiplication for the `MixColumns` step.

2.1.5 Security measures

Implementing algorithms as a sequence of arrays to read through is required to consider the white-box model, but not sufficient. The tables related to the key all contain secret information in them, that anyone can read. The use of internal encodings tries to tackle this issue. They are random bijections merged with operation tables, to achieve confusion. In order to remain compliant with specifications, the application of such functions is done so that when they are positioned after a legitimate operation, they are canceled by another positioned before the next one. This way, the intermediate values between arrays are impacted by these encodings. Formally speaking, if f and g are two operations, and if a cryptography standard stipulates that they should be used in chain, as $g \circ f$, then a random bijection h is defined, and applied to it in order to get $g \circ h^{-1}$ on one hand, and $h \circ f$ on the other. The precalculated tables will be modified, but not the general result, as $g \circ h^{-1} \circ h \circ f = g \circ f$. A specific table can be merged with multiple encodings on both of its ends. This chaining process is illustrated with three functions in Figure 2.2. A considerable advantage of this technique is that it has no cost at runtime, meaning it does not imply any additional computation: the same number of arrays will be read. However it implies a lot of random generation during the creation of the executable, but this is usually done in an industrial context, with high-performances, completely controlled computers.

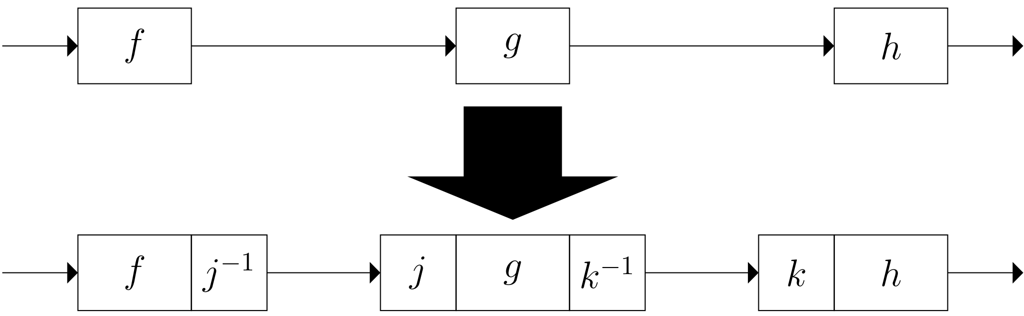


Figure 2.2: Illustration of the chaining of three operations, before (top line) and after (bottom line) inserting encodings and their inverse between them, but without influencing the global output.

Once *confusion* has been added with encodings, *diffusion* must be, too. For this, linear transformations are used, in the same way as internal encodings. That is, they are merged with tables, after and before them, while making sure they will all cancel themselves in order to stay respectful of the original design. These functions need to be linear in order to provide the desired bit dependency: a modification on any bit of the input should impact the highest number of bits in the output. Besides being obviously random, they also need to be invertible so that they can get canceled.

Finally, and optionally, external encodings can be applied. Just as internal encodings, they are random bijections, however they are located at the beginning and/or the end of the whole algorithm. Their purpose is to have an executable that generates or works from encoded data. Ciphering would not only encrypt a block, it would also encode it so that only actors knowing the bijections could recover it. Most importantly, it can be the case when deciphering too, allowing to get the benefits of a state-of-the-art standard, with an additional measure to protect the cleartext. While this technique implies to abandon the specification compliance (as the white-box implementation will not generate the same output as a classic one for the same input), it is considered very useful in the field of DRM, where it is common to want to restrict the users able to exploit some particular data.

2.1.6 The limits of encodings and linear transformations

Once these techniques have been applied, tables are deemed to be locally secure, meaning when considered individually, they cannot leak secret information. However, as attackers have access to everything in the white-box model, they are not limited to examine only one particular array: they can consider their interactions, and the whole executable. Consequently, a flaw in the specification design can also cause trouble in a white-box implementation anyway. The general design and measures implemented in such an executable, despite the followed specification, can also generate the leak of secrets. Thus, the supposed perfect security of tables is not relevant as their exploitation and use can be faulty.

2.1.7 Expectations around white-box implementations

In the white-box scenario, the expectations around it and the very definition of safety are not precise. Emphasis is often given to the robustness of such implementations, but does it matter if one can copy these executables and simply execute them elsewhere? Why focus on the secret key when the software covering it can be moved? This action, called code lifting, complicates the goal settings of developers. Also, while it is commonly considered to deploy white-boxes for decryption, their purpose is not clear either: storing both a ciphertext and the executable capable on deciphering it on the same device necessarily implies the risk of an attacker using the software to process the data. Delerablée *et al.* works [DLPR13], in 2013, give a formal definition of the expectations around white-box cryptography, specifying three characteristics:

- **one-wayness**: it should not be possible to run an implementation in reverse. More precisely, a binary made for ciphering should not allow deciphering, and vice versa.

- **incompressibility**: such executables should not be able to be compressed. Their weight are part of the protection they offer, as being harder to store and share than a 128-bits key is a feature.

- **traceability**: when an instance of a white-box executable is marked uniquely in order to be traceable, it should not be possible to remove this fingerprint.

2.1.8 The ephemeral security of white-box implementations against black-box equivalents

In all cases, white-boxes implementations are recognized as less robust as the equivalences based on the black-box security model. While the latter, even if possibly flawed [BLLL15, BKL+13], aims for implementations to last over the very long term, the former are deemed to hold weeks at best, as shown by contests dedicated to measure their resistance [BT20]. Using cryptographic implementations with temporary security guarantees implies many constraint when considering their use on actual production systems. The main one is a regular rotation of all sensitive material. If a particular white-box instance is estimated to stay secure for a certain period of time, then it should be considered obsolete at the end of a shorter delay, and a new one should be provided. This mode of operation is, among others, exhibited in the internals of SafetyNet by Google [Rom22]. In the industry, such attempts to measure the robustness of an implementation are usually done through penetration tests, often conducted by qualified actors. It is thus evaluated as a delay in days, weeks, or months.

The ephemeral security of white-box implementations is mostly explained by the considerable number of attacks that have been developed against them. Since its birth, this security model was targeted with various attacks, notably based on cryptanalysis, like the Billet-Gilbert-Ech-Chatbi (BGE) one [BGE05]. Shortly after these offensives, many more relying on cryptanalyses were developed, on DES implementations with [GMQ07] and [WMGP07], on AES ones with [DMWP10], or on both kinds with [MGH09]. However, the following years have witnessed the emergence of attacks that are particularly effective. In [DMRP13], De Mulder *et al.* present a new cryptanalysis on white-box AES implementations. The BGE attack was enhanced in [TMM+13].

2.1.9 Hardware attacks and their migration to the white-box model

Hardware attacks are a set of offensive methods based on the physical properties and environment of a target. While initially intended for hardware components like SoCs, they can also make sense in a virtual context. The two main categories of such attacks rely on side-channels and fault injection.

System considered secured can sometimes present compromising information in unexpected ways, against their will, even when considering black-box security models. These ways are called side-channels. Most of the time, they are either based on power consumption (allowing one to infer secrets by probing it) [HTG+11, JKL21], time (allowing one to obtain data according to delays showcased by the system) [YGH17, KJFK18], or electromagnetic emissions (allowing one to understand data by capturing them) [CPM+18, ZK19]. Once these gave someone information considered unattainable, the security is compromised.

Attacks based on fault injections also represent a similar threat, even to systems designed as black boxes too. A fault injection is a perturbation induced on a component, resulting in an originally undesired modification of the memory or the execution. The change in data or in the control flow of a program can lead either to mathematical leaks or in undefined behaviors, both possibly leading to security compromises. Indeed, even if an attacker still cannot observe the insides of the system, the injection of a fault can modify the outputs which can be easily read by him. In practice, fault injection can be done using electromagnetic emissions, lasers, or by controlling the power source. Effective instances of attacks based on fault injections are the Differential Fault Analysis (DFA) [BS97, DLV03b] and the Persistent Fault Analysis (PFA) [CB19, FXX+18].

Side-channels and perturbations usually require hardware tools to be exploited, however they can also be modeled in a purely software way. This is frequently done in order to prototype an attack before actually mobilizing heavy equipment. Software instrumentation tools like Rainbow [Led] or QBDI [Qua] can be deployed to estimate metrics about power, time, or

memory, and to simulate the injection of a fault. This way, it is faster, easier and cheaper to evaluate a potential attack. The porting of hardware attacks to software is impactful against the white-box model: using the same methods, one can try to find side-channels around such an implementation, or to perturb it.

In fact, side-channels and fault injections revealed themselves to be particularly appropriate in the white-box model. As soon as 2003, [JBF03] showed how perturbing implementations of this kind could achieve convincing results. An important step was the porting of the DFA attack [DLV03a] to the white-box model, presented in [ECJ15]. Similarly, another powerful method was elaborated at this time, called the Differential Computation Analysis (DCA) [BHMT15], based on the Differential Power Analysis (DPA) [KJJR11]. It was detailed in [BBMT18] three years after being disclosed.

2.1.10 Reverse engineering and analysis circumvention through simulated hardware attacks

Some of the attacks targeting white-box implementations, notably the DCA and DFA have one considerable advantage: they require little to no study of the targeted binaries. In other terms, they can provide a way to circumvent reverse engineering. However, the difficulty of understanding the executables is a foundation in which rely white-box implementations. Indeed, retro engineering is complex, it also requires knowledge, and thus experimented people. The end result for any organization is that it costs large amounts of time and money, making it unaffordable. This situation is an integral part of the security model, and is among other reasons why the security of white-boxes is measured in time. In consequence, any developer of such software will inevitably mainly focus on avoiding any of these major risks.

Thus, providers of cryptographic implementations running in open contexts need to prevent many things from happening, from binary instrumentation to execution in virtual machines, including also data and code alteration, environment manipulation in general, and retro engineering. These tasks are each demanding on their own, and require large workload individually, aside from the main, heavy challenge that is the implementation of cryptography. To help in this situation, and to ease development, many layers of protection are exploited. This is why recent products tend to embed multiple, different technologies to cover these scenarios, such as code obfuscation both in static or in dynamic ways, integrity control, rooting status check, and instrumentation or virtual machines detection, between others. White-box implementations are no longer monolithic binaries only executing cryptography, they also deal with many runtime risks in parallel.

2.1.11 Asymmetrical white-box implementations

Historically, this field of research only focused on symmetrical cryptography, where all keys are private. The interest for asymmetrical schemes in academia is relatively new. In 2018, Zhang *et al.* introduced an implementation [ZHH+18] complying with the signature scheme from Institute of Electrical and Electronics Engineers (IEEE)'s P1363 standard [IEE13]. Another work on identity-based signatures was conducted soon after in [FHW+20]. Barthelemy proposed a white-box design based on lattices in 2020 [Bar20]. Finally, an implementation of the Hidden Field Equations (HFE) signature scheme for open environments was disclosed in 2022 [GG22].

In fact, white-box products proposing asymmetrical schemes are present in the industry for quite some time: after some investigating, one will quickly find promotions for Rivest–Shamir–Adleman (RSA) implementations with this security model. However, adapting this algorithm for such a context is known to be extremely complex. The classic precomputation technique cannot be applied directly, as RSA requires to work with very large numbers, and methods similar to the one used for the `MixColumns` step in AES do not apply. Hence why to this day, the few known white-box RSA implementations like [CMTE22] are not up to the industry's standards.

Since the promoted commercial products naturally do not disclose how their internals work, it is impossible to know how their developers proceeded. Most likely, these implementations do not exploit traditional white-box methods, and rather rely on heavy common obfuscation techniques.

The same cannot be said about the Elliptic Curve Digital Signature Algorithm (ECDSA). It was much more investigated and considered as a candidate for white-box implementations. In [ZBJV20], Zhou *et al.* propose a methodology for developing one, but it requires a trusted, distant third party. The encountered difficulties when trying to achieve this are explored in [DGH21]. The 2021 WhibOx contest stimulated the research on this subject, by allowing the submissions of implementations, and exposing them in order to observe how they would resist to attacks. Defensive and offensives methodologies exhibited in it are both analyzed and described in [BDG+22] and in [BBD+22]. This event showed that white-box ECDSA implementations could not be considered in an industrial context, as no candidate lasted longer than two days.

2.1.12 Exploitation in a defense in depth approach

Initially intended for exposed and open environments, the white-box model is now also considered for other contexts. Indeed, its common techniques and methods can be deployed to software running on HSMs, TEEs and SEs, too. While this might appear useless since, in this case, the private material would be protected by the hardware features, it can be seen as a way to enforce a defense in depth. In this paradigm, securization is considered in every layer and module, instead of only being applied to a single line of defense. Hence, if a security vulnerability would happen to be found in a hardware solution, the software tailored for the white-box security model would add an additional coverage. This frame of mind is now widespread in the payment industry, where it is more and more considered during certification processes.

2.2 Confidentiality of processes in Android

A common target for software developed with the white-box security model in mind is Android. It is an OS tailored for smartphones. It is developed and maintained since the 2000s, and was bought by Google in 2005. Soon after its creation, and before its first release in 2008, it was decided to base it on the Linux kernel, making it what some would refer as a Linux distribution. Unlike most of them, it embeds many optimizations and tweaking specifically dedicated to portable devices, particularly phones. These optimizations favor performances and power consumption. Besides, Android also differs from regular Linux distributions due to many conception choices.

2.2.1 Abstraction choices

Designing a complex software system in charge of hosting third party binaries is difficult. Any layer of abstraction, while acting as an additional safeguard, can also be a weakest link if it creates attack vectors. Thus, from a security point of view, the rising of an abstraction stack is not an all win situation. For instance, the introduction of Memory Management Units (MMUs) made it so executables do not manipulate raw memory addresses: they can only consider their own virtual space, preventing them from interfering with the data of other processes. However, relying heavily on this feature in a security model can be dangerous: [SCHK18] shows that their address translations can be subject to cache timing attacks, and [BKE+17] demonstrates how MMUs can help defeating another protection, called Address Space Layout Randomization (ASLR). Another abstraction layer exhibiting the same phenomenon, popular due to its flexibility, portability and the performances it can achieve, is Just-In-Time (JIT) compilation. Very common in web

browsers with JavaScript engines, it however requires to be able to map memory regions that are both writable and executable, which is quite hazardous [GH18, CTu22]. On iOS, such memory mapping is not allowed to third party products [JIT24].

Like a lot of other OSs, Android is made to run executables from many sources. Such contents are called applications. They are essentially developed using the Java language or the Kotlin language. However the resulting bytecode is not directly interpreted: it is first converted to one based on registers rather than on stacks. Since version 5 and the switch to a new runtime environment called Android RunTime (ART) in 2014, compilation to native instructions is essentially done Ahead-Of-Time (AOT), meaning before execution, during installation, and not JIT. The description of native code in the sources of a program is possible, for example by exploiting the Java Native Interface (JNI), as usual. However this is mostly done for high-performance use cases like video games, and sometimes to improve obfuscation, as it is easier to implement on native binaries than on bytecode.

2.2.2 Applications supply chain and moderation

While applications can potentially come from anywhere, Google puts continuous efforts to concentrate their provision on the shoulders of the Play Store, its official distribution service. Controlling the executable supply chain of a platform is considered crucial for its security, as it allows to analyze, filter and moderate the downloadable contents. The main goal is to prevent people from installing applications that are malicious or undesirable. This qualification regroups different threats:

- **Applications dangerous for the system.** They could, for example, sabotage it by overexploiting resources like memories, network, processing power... Or they simply could be malfunctioning or not completely compatible with Android systems.
- **Applications dangerous for users.** They mostly represent a menace to their privacy, as they could try to get access to sensitive data for illegitimate purposes. Another flaw they could exhibit is unregulated monetization, via unsupervised advertising for instance.
- **Applications considered unwelcomed by Google following moral or cultural standards.**

The third point is outside the scope of this thesis and will not be addressed here. The others, which have a more technical aspect, are essential to understand when considering the security of Android.

2.2.3 Threats distribution over the several, distinct abstraction layers

The number of different security threats an application can cause is high, and they take place at different levels of abstraction of the system. Incidentally, their respective countermeasures will be located at various points.

- Protections against file systems or device files violations are enforced at the Linux kernel level, simply using Access-Control Lists (ACLs), or with the help of Security-Enhanced Linux (SELinux), deployed in Android systems progressively since version 4.3. This is the most logical way of applying them since permissions for each files are defined by system designers, and users rarely need tweak them.
- Protections against violation of a lot of hardware resources and OS features are enforced at the level of the Android framework, mostly implemented in Java. This is the most adapted way for this, as the users is often solicited to know if an application should be able to access the camera, the network, the contact list, or notifications, for example, and this is done using the high-level User Interface (UI).

- Finally, Google tries to detect privacy violations or resources abuses before installation, as soon as an application is submitted to the Play store. This works because these threats potentially concern every devices and do not depend on the users' configuration, thus it is possible to infer no one has any interest of even being in contact with these executables. However they are millions of applications on this distribution platform. Since human verification and study over such a large number of executables is unthinkable (even more when considering each updates), most of it is automated. While this makes the process of moderation a lot less expensive, it is however not perfect and actually proves to be a heavy, complex task: many malicious applications can fall between the cracks [Git20].

Storage encryption, whether it is Full-Disk Encryption (FDE) (since version 4.4) or File-Based Encryption (FBE) (since version 7.0), helps protecting data at rest in case of a device theft for instance, but is not exploited to regulate applications accesses at runtime: once a user has unlocked the master key, the OS would decrypt a file if an application managed to make an unauthorized request through the aforementioned protections such as ACLs or SELinux. The different abstraction levels facing risks and providing countermeasures are illustrated in Figure 2.3.

2.2.4 Isolation breach in practice

The considered threat categories are often not exclusive. A breach can exploit misconfiguration both on the native level and the Android environment, along with a false negative from the Play store. Applications can be both hazardous to the system and the user: recovering a secret protecting the two of them can allow to damage both the device and the user's privacy. For instance, recovering a PIN code used to unlock a phone provides the opportunity to remove many contents and to get access to personal messages and information.

There have been many attempts to improve the isolation between contents and application in Android. Some of these are based on processors' features, like virtualization [Ave17], or the multiplicity of cores in today's embedded architectures [RRH15]. However most efforts rely on software mechanisms. Models based on frameworks added to the OS are explored in both [BDD+11] and [PZZQ16]. In [DMVN15], Das *et al.* propose a paradigm based on roles, allowing several instances of the same applications to be isolated. A more classical way for containerization is explored in [XLL+15], using kernel features such as namespaces and cgroups. In all cases, the Bring Your Own Device (BYOD) use case is a direct target. This scenario is common and widespread in the professional world: it consists on having companies and their human resources exploiting personal machines, such as the employees' phones and computers. Content distinction thus become vital. A promoted solution for this context is the KNOX technology from Samsung, which gathers multiple features from multiple layers, and is analyzed in [KW16]. Besides, in [AKPK16], Athanasopoulos *et al.* explore the protection of applications from the very native libraries they are using, and which are thus running in the same memory context as them.

2.2.5 Code entry spying

Among the many risks that are implied by unwanted interactions between contents, an emblematic one is the spying of password or PIN code entries. As this process often serves for authentication, a successful attempt can result in usurpation or theft. PIN codes, in particular, are especially targeted as their set of available possibilities is not large, whereas they are used in many sensitive domains, such as banking, or telecommunications.

Attacks on PIN codes can happen on many layers. It can be done in the user space, which is the case when one is exploiting a non constant-time verification. If one get access to kernel

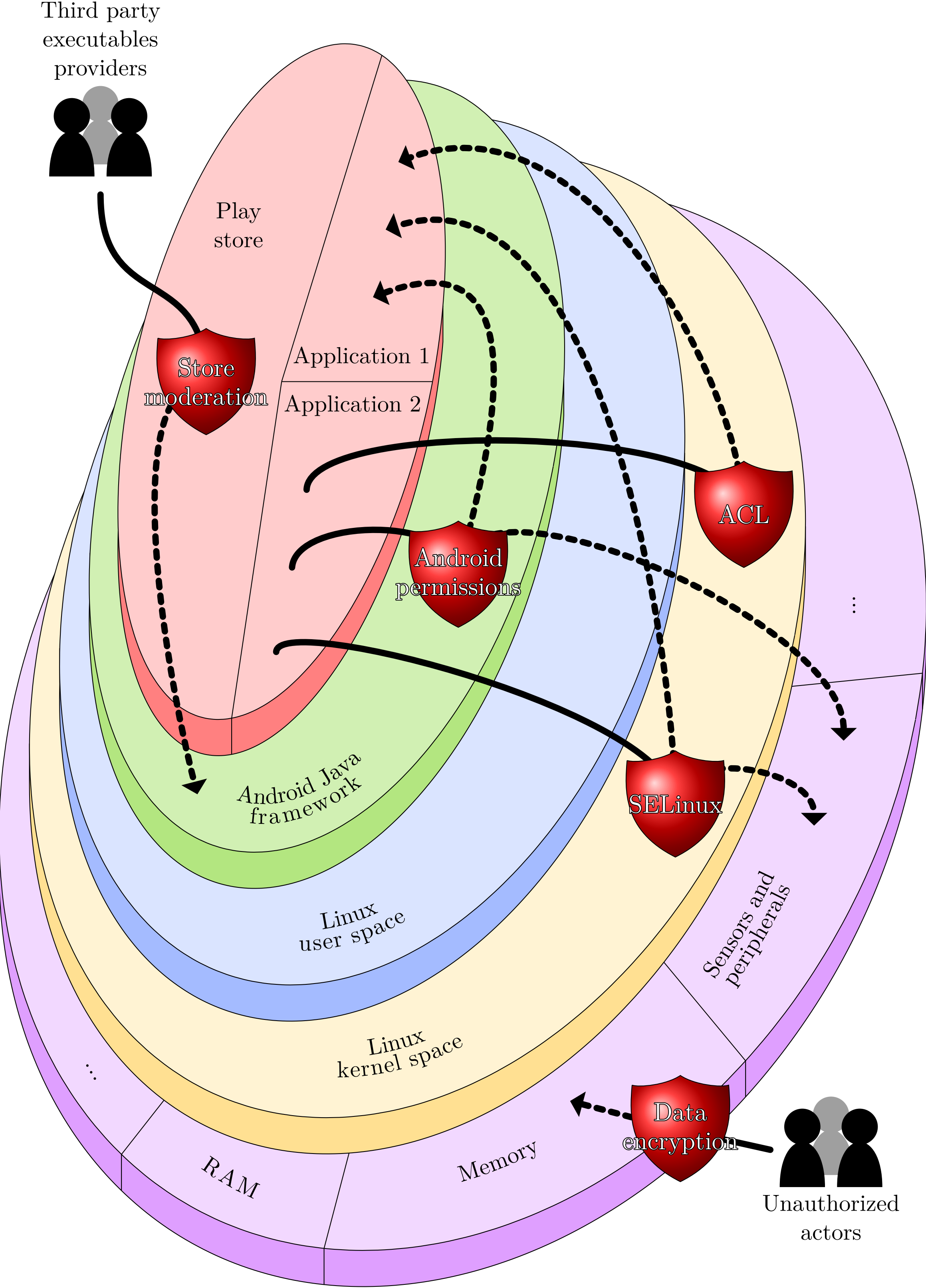


Figure 2.3: Depiction of the different levels of abstraction at which threats and their countermeasures take place. Full lines represent potential malicious accesses attempts, dotted lines represent which accesses would be successful if the corresponding countermeasure, symbolized with a shield, would not exist.

space, it can obviously be done there too. More importantly, it is often done outside of the system in itself. To achieve this, side-channels are frequently exploited. The most studied one are sounds [PLHQ20, CCBG19], ultrasounds [LLD21], motions [CC11, XBZ12], power [CGYW21], time [FKK10], or a mix of many of these [BJB17]. Traces on keyboards, possibly as chemicals [NP03] are also a possibility.

Spying can be done when code entry is ongoing on many different devices: ATMs, payment terminals, digital door pads at the entrance of buildings, lockers, and more. In the works presented here, a focus will be undertaken on Android: the concentration of contents and services on personal phones makes them an increasingly important target, and they are the ones which will more and more manage PIN codes and passwords. Also, as attested by the SPoC standard in 2018, their relatively new role at accepting codes entries is official now, at least in the payment industry.

The notion of Android personal devices as trusted PIN entry pads can be impacted by a considerable obstacle: the many issues raised with its permission system at the Java framework level. It is, in principle, quite similar to capabilities: instead of generalizing ACLs to every resource, it focuses on runtime specific actions: accessing contacts, the camera, or the network, being able run in the background, to have processes awake in sleep mode... Android users are familiar with them due to the frequent pop-up menus they initiate, however these are not always a requirement depending on the security policy. The shortcomings of this system has been explored with a solver approach in [Jé23]. In [BYW17], Bai *et al.* propose a way to limit inferences based on sensor data by reinforcing the permission system using static instrumentation. Finally, the custom permission feature has been analyzed in [LDL+21]: above the system permissions, custom permissions can be defined and declared by applications to share their contents and functions with other processes.

Synthesis

In this chapter, the state of the art regarding major aspects when considering security on COTS products is explored. The first one is white-box cryptography. Its history and technical bases are described. The subjects of expectations around it and its limits in reality are addressed. The new features and uses they can nonetheless provide are discussed.

The second aspect is the confidentiality of processes in the Android OS. The technical abstractions choices implemented in it were detailed, along with the risks applications can potentially bring. A summary of how threats can impact different layers and points of the system is given. Finally, the chapter focuses on the consequences they can have on isolation between contents, and details this concern with one concrete example: code entry spying.

Part II

Contribution

Chapter

3 Offensive and white-box propositions for the McEliece cryptosystem

Abstract

In the exploration of secure operations in untrusted environments, the McEliece cryptosystem reveals itself to be worthy of interest. Indeed, it is known to be quite robust, is flexible as an asymmetrical algorithm, can provide good performances, and has good, recognized post-quantum capacities. As such, solution providers can be tempted to implement it in uncontrolled environments, especially ones that are not quite powerful, like smartphones. The definition of this cryptosystem is thus recalled. All its features, characteristics, and legacy are also presented and detailed.

Then, an offensive work against it is presented, using the injection of a fault. The secret key in the McEliece cryptosystem was never successfully targeted by attacks based on fault injection, as the error correction capacity of it makes it profoundly complex. The attack described here will thus focus on implementations rather than on the original specification. In particular, a focus is made on the ones developed for the Arm architecture. It is shown here how, with a single bit flip, the replacement of a particular instruction, exhibited by a common way of depermuting data at the beginning of decryption, can lead to a recovery of the permutation matrix, which allows the recovery of the secret key in general. Detailed examples of specific scenarios are provided for better understanding about the different methodologies that can be used.

In the face of these risks, the question of how to protect these algorithms is addressed. Specifically, in an uncontrolled environment, fault injections can be reproduced easily without any cost, by using any type of binary instrumentation. Sometimes, they even allow the circumvention of reverse-engineering. Therefore, some propositions that can be considered and studied about a white-box implementation of the McEliece cryptosystem are finally provided, that rely not simply on general obfuscation, but on actual precalculation along with encodings and bijections, as often in this model.

Scientific valorization and recognition

This work led to acceptances from the following events, journals or institutes:

- *Journée thématique sur les Attaques par Injection de Fautes* (JAIF) 2022, for: *L'attaque en faute : la bête noire des boîtes blanches*
- *Security of Software/Hardware Interfaces* (SILM) 2023, at *European Symposium on Security and Privacy* (EuroS&P), for: *Faulting original McEliece's implementations is possible — How to mitigate this risk ?*
- Patent application number WO2024083855 filed at the *Institut National de la Propriété Industrielle* (INPI) on October 17, 2022: *Clés cryptographiques en boîte blanche*

Chapter content

3.1 The McEliece cryptosystem . . . 30
3.1.1 Interests . . . 30
3.1.2 Definition . . . 30
3.1.3 Performances and footprint . . . 31
3.1.4 Legacy . . . 32
3.2 Offensive . . . 32
3.2.1 Cryptanalysis history . . . 32
3.2.2 Hardware attacks history . . . 33
3.2.3 Motivations for offensive research . . . 34
3.2.4 Preliminary inspections . . . 34
3.2.5 Determining hot spots . . . 34
3.2.6 The permutation matrix in the McEliece cryptosystem . . . 35
3.2.7 Exploiting the Arm instruction set . . . 37
3.2.8 Information extraction tactic . . . 38
3.2.9 Operands order explanation and determination . . . 42
3.2.10 Practical example when $accumulator = accumulator - matrix\ line$ . . . 43
3.2.11 Practical example when $accumulator = matrix\ line - accumulator$ . . . 47
3.2.12 Resulting metrics . . . 49
3.3 White-box measures . . . 50
3.3.1 Context and motivations . . . 50
3.3.2 Decryption . . . 51
3.3.3 Encryption . . . 53

3.1 The McEliece cryptosystem

3.1.1 Interests

A year after the public release of RSA [RSA78], Robert J. McEliece revealed his eponymous cryptosystem in 1978 [McE78]. For the first time, this algorithm allows to make a trapdoor function out of error-correcting codes. In overall, it is heavily based on linear operations. While the original specification prescribed Goppa codes [Val70], an user willing to accept the consequences can in fact use other linear codes.

While designers and developers favored RSA for long, the McEliece cryptosystem has become relevant again for several reasons developed hereafter. First, it is a longstanding algorithm and thus benefits from decades of scrutiny, implying a good reputation. Its mathematical properties allow it to achieve extensively high performances. It went on to become a considerable choice against potential attacks conducted by quantum computers. Like all asymmetrical algorithms, it provides flexibility when designing systems for varied use cases. Finally, it exhibits characteristics that can help envisaging a white-box version of it.

3.1.2 Definition

The McEliece cryptosystem is an asymmetrical cryptographic algorithm, oriented towards Key Encapsulation Mechanisms (KEMs): encryption is done with the public key. To generate a pair of keys, one must first consider a finite field $GF(2^m)$ where m is an arbitrary strictly positive integer leading to $n = 2^m$, the length of codes in bits. The user also has to choose k, the

length of data blocks in bits, and t, the maximum number of errors the code can correct, so that $\frac{n-k}{t} \leq m$.

Next, a random irreducible polynomial of degree t has to be chosen in $GF(2^m)$. The $k \times n$ matrix G generating the associated Goppa code must be constructed. Additionally, one must also randomly generate the $k \times k$ scrambling matrix S, which is nonsingular in $GF(2)$, and the $n \times n$ permutation matrix P, in the same field. These three matrices S, G and P, considered distinctly, are the private key in the cryptosystem. The public key, G', is the multiplication of these, such as $G' = S \times G \times P$.

To encrypt a data block, one must multiply it with the G' matrix. This will scramble, encode, and permute the data all in once. Then, the user should create a random n bits error vector with t as a Hamming weight. The encrypted block is the addition of it with the preceding result. Thus, if x is the cleartext, then y, the ciphertext, is defined as $y = xG' + z$, z being the error vector. This algorithm is represented in Figure 3.1.

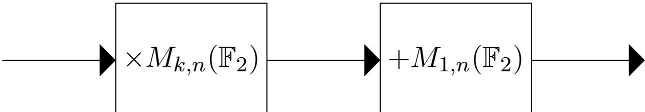


Figure 3.1: Diagram representing the encryption in the McEliece cryptosystem.

To decrypt a block, one must multiply it with P^{-1}. Only then, he will be able to use a fast decoding algorithm on it, such as the Berlekamp-Massey or Patterson ones. Finally, the data can be unscrambled by multiplying it with S^{-1}. This algorithm is represented in Figure 3.2.

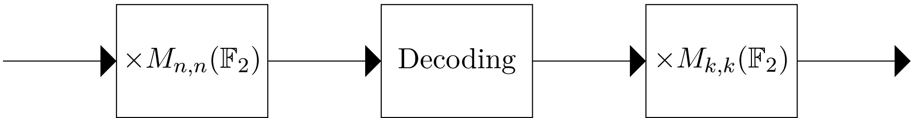


Figure 3.2: Diagram representing the decryption in the McEliece cryptosystem.

The McEliece cryptosystem works because permutations are operations that maintain the Hamming weight of the parameter: the output necessarily has the same number of bit set than the input. Thus, when adding the right quantity of error to a codeword after it has been permuted, one is assured that, even when it will be depermuted, it will not be over corrupted. However, only depermuting to decode it after is only possible when having the P and G matrices distinctly, separated, hence the impossibility the decrypt using only the public key, which is a single matrix associated with a strong linear injection. An attacker without the private key cannot decrypt a block, because the permutation matrix applied during the encryption will have hidden the structure of the code used. An exhaustive search for the original code G is, of course, not viable as soon as the users have chosen proper parameters. Besides, the S and P matrices would also need to be determined. In the same spirit, an attacker could disregard that $G \neq G'$ and try to find the nearest codeword nevertheless. However, the Syndrome Decoding Problem (SDP) has been proved to be NP-hard for linear codes [BMH78]. Using Information Set Decoding (ISD) is a possibility in order to be more efficient, as explained in [CC98] and [CS98]. It has also been practically explored in [NFK21]. Nonetheless, these are easily discarded by using long enough codes.

3.1.3 Performances and footprint

In the first years of asymmetrical cryptography, RSA was widely privileged over the McEliece cryptosystem, since it requires smaller keys. Indeed, the former used private keys around a few hundred bytes, while the latter's private keys are largely over 100 kilobytes. However, with hardware improvements and the vast decrease in the price of memory, this constraint is less

burdensome today as it used to be. Nonetheless, it remains important to note that the size of the keys grows squared with respect to the parameters, as they are based on two-dimensional matrices.

Many attempts were made to gain better performances or shorter keys by implementing changes around the error-correcting codes. This requires few modifications from the original specification. Despite their benefits, these alterations often have the downside of introducing security flaws. Multiple instances of such cases are listed in subsection 3.2.1. Because of these, the classic, binary Goppa codes are seen as a safe choice by system designers and developers.

Since the McEliece cryptosystem is heavily based on linear operations that can be modeled as matrix multiplications, it exhibits a good performance potential, and it is easy to optimize its execution speed, or to accelerate it using hardware features. This can be done by exploiting the Single Instruction Multiple Data (SIMD) instruction set extensions provided by the processor if any, or by operating the GPU if the platform embeds one [EFK16].

3.1.4 Legacy

Despite being published in 1978, the McEliece Public Key Cryptosystem (PKC) generated a vast scientific heritage for the next decades, and research is still active on the subject today. Adjustments were proposed to add features. For example, in [CFS01], Courtois *et al.* show how one can use the Niederreiter variant in order to provide digital signatures, as they are not possible in the original specification.

Various progress in the field on quantum computing started, at some point, to impact the cryptography community and research. As RSA was recognized as vulnerable against attacks conducted by such calcultators, the necessity to provide new, resistant asymmetrical algorithms became urgent. As a first step, various institutions made official communications to give general guidance [ANS22, Bun20] or recommendations about what primitives to use until final alternatives are developed. In particular, PQCRYPTO, a forum gathering expert researchers on the subject, was mandated to emit guidelines about how to deploy encryption and signature systems in this situation. In 2015, they officially recommended to use the McEliece PKC to do asymmetrical, public-key encryption [DWTN15].

In 2016, the National Institute of Standards and Technology (NIST) initiated a program to bring a long-term standardization of quantum-secure algorithms in order to do encryption and signatures [Nat17]. Only asymmetrical specifications were considered, as attacks from quantum calculators are not as compromising for symmetrical ones. 59 schemes were submitted for encryption, while 23 were for signatures. One of the former is Classic McEliece [ABC+22]. While outside of the scope of the works presented here, it is nonetheless heavily based on the original McEliece specification. It is also the subject of many offensive and defensive works on its own. This candidate made it to the fourth round [Inf22], as it provides robustness in the post-quantum context, notably due to the exploitation of error-correcting codes.

The debate around the proper key size to use has been vivid since the creation of the McEliece PKC. The original specification took $n = 1024$ as reference. Following experimental results, Bernstein *et al.* recommended in 2008 the use of $n = 2928$ to achieve 128-bit security, in [BLP08]. However, hardware capabilities greatly evolved since then, and new offensive quantum methods were developed since. The initial recommendation from the PQCRYPTO forum advised in 2015 to use $n = 6960$ for the same level of security [DWTN15].

3.2 Offensive

3.2.1 Cryptanalysis history

If the parameters of a specific implementation of the McEliece cryptosystem are properly chosen, then a malicious actor cannot consider brute-force methods against it, whether it is to find the

original code G, or to find a nearest codeword, even with the help of ISD. The only remaining option would be to recover the structure of the code in place, thus allowing to have a fast, affordable decoding algorithm. However, the original proposal for the McEliece cryptosystem relies on binary Goppa codes, which so far remain one of the few families of codes to have largely resisted attempts at devising such attacks. Others, which might allow for smaller keys or better performances, can also bring cryptographic flaws:

- In [Har86], Niederreiter proposes the use of Generalized Reed-Solomon (GRS) codes to get smaller keys. However, Sidelnikov and Shestakov exposed in [SS92] an attack with a complexity of $\mathcal{O}(n^3)$.
- In order to pursue the same objective, Sidelnikov unveiled in [Sid94] a version using Reed-Muller codes. But Minder and Shokrollahi have developed a structural attack against it, explained in [MS07].
- Janwa and Moreno described in [JM96] a proposition exploiting algebraic-geometric codes. However, this idea was countered by both Faure and Minder in [FM08], along with Couvreur *et al.* in [CMP14].
- In [Gab05], Gaborit uses a variant based on Bose–Chaudhuri–Hocquenghem (BCH) codes. But Otmani *et al.* show it is not secure in [OTD10].
- Berger *et al.* attempted to use quasi-cyclic alternant codes in [BCGO09], just as Misoczki *et al.* tried to exploit quasi-dyadic alternant codes in [MB09]. However, both attempts were proven flawed by Faugère *et al.* in [FOPT10].
- Bernstein developed the idea to use codes over $\mathbb{F}_q$ instead of $\mathbb{F}_2$ in Wild McEliece [BLP11], however this was attacked by Couvreur *et al.* in [COT17].

In the context of quantum computing, the McEliece cryptosystem is deemed to be secure against attacks based on Shor's algorithm [Sho94]. However, because of Grover's algorithm [Gro96, Gro97], users might need to implement bigger keys to be safe [Ber10]. These elements helped to make the McEliece PKC deemed viable in such an attack model.

3.2.2 Hardware attacks history

Several attacks based on side-channels have been developed against the McEliece cryptosystem. Two main metrics are used. Firstly, time has been exploited on the execution of the decoding algorithm to recover the secret error vector [SSMS10, Str11a, STM+08], the secret permutation [Str10], the secret Goppa's polynomial [AHPT10], or the secret code's support [Str11b]. Secondly, power consumption has also been exploited, in order gain information on on the secret error vector [MSSS11], the parity check matrix [CEVMS14, VMG14], or both the parity check matrix and the permutation one [HMP10].

Potential attacks based on fault injection were investigated against the McEliece cryptosystem. However, none managed to demonstrate vulnerabilities on the private key. In [CD10], Cayrel *et al.* show how the public key can be compromised using this type of methods. However, they conclude the McEliece cryptosystem is resistant due to its capacity to correct faults, regardless of their origins. Regarding the secret key, they declare to be working on exploiting fault injections against it, but never reported results since 2010. Such attacks were only successful on more recent specifications based on, but different from the original PKC, whether they focus on recovering the message [CCD+20] or the key [DK20, PGD+22].

3.2.3 Motivations for offensive research

As explained, no attack based on fault injection was developed and publicly disclosed against the secret key in the McEliece cryptosystem to the best of our knowledge. It is paramount to evaluate if it is because of the specification's robustness and because of the error correction, or because of the general underevaluation of this risk. And, if the possibilities of such attack vectors exist, they must be investigated thoroughly, because as it was detailed in subsection 2.1.10, they can have significant consequences, like, among other things, to circumvent advanced safeguards and technologies dedicated to obfuscation and protection of products living in the REE. The stakes for companies are thus critical.

Despite the age of the McEliece cryptosystem, the context around today's industries gives it some attractiveness. It being still considered viable despite all its years makes it look robust, and the fact it proposes asymmetrical cryptography is handy for system designing. It exhibits good resistance in quantum attacks and is particularly appreciated in telecommunications thanks to its use of error-correcting codes. All these naturally make it an important candidate when conceiving secure systems for smartphones, especially when preparing a transition to newer algorithms developed specifically for the quantum threat. Thus an offensive analysis around it is worthy in this framework of inspecting the security possibilities in personal, uncontrolled environments. Eventually, the specifics of such environments will have to be taken into account: they are almost always based on the Arm architecture, are very energy efficient, and host many different contents and executables.

3.2.4 Preliminary inspections

A major interrogation appearing when undertaking an offensive approach with fault injections against the McEliece cryptosystem is deciding if it is better to aim for the specification or the implementations. The former implies to find flaws in the standard as described by Robert J. McEliece himself in 1978, while the latter rather implies to find defects in implementations of the algorithm. A successful attempt at achieving the former seems unlikely: this cryptosystem has been existing for almost half a century, and no team has been able to describe a method for retrieving data on the secret key using the injection of faults. As recalled in subsection 3.2.2, only the public key has been compromised, and only variations of the McEliece cryptosystem let secret key information recovery achievable. In comparison, targeting implementations seems much more fruitful: they can introduce a lot of loopholes, whether it is because of careless work by a programmer, or because of ignored, undocumented or apparently irrelevant characteristics exhibited by the hardware or software environment that can actually be maliciously exploited.

Then, it makes perfect sense to specifically target implementations of the McEliece cryptosystem for the Arm architecture, since smartphones are the COTS products receiving the most attention, and they almost all are based on it. When focusing on an architecture, especially from a fault injection point of view, the area most likely to be of interest is the instruction set. It acts as the interface between the software and the hardware, and while the latter has much freedom in how it implements each operation, it still has to abide by the respective instructions as specified. In particular, each commanded action is designated through an operation code: an arbitrary, specific value dedicated to it. When a compiler is working, its correct picking of instructions is essential. If the operation code of an instruction from an executable is modified, the global behavior can be radically changed. In this way, investigating the consequences of faulting operation codes in implementations of the McEliece cryptosystem becomes highly relevant.

3.2.5 Determining hot spots

From the point of view of an attacker, the goal is to target the parts of the McEliece cryptosystem which are best able to provide information about the key, in the greatest quantity possible, as

effortless as possible. With this mindset, the three parts composing a secret key that can be considered are:

- **The scrambling matrix S.** It simply acts as a linear invertible transformation applying diffusion. It might seem like a reachable target, as it is located directly at the beginning of the encryption, and at the end of the decryption processes. However, even if it was to be completely recovered, it would not be sufficient on its own: the collaboration of the permutation and the correcting code is what achieve the essential cryptographic work, and there are no direct link between them and the scrambling matrix.

- **The error-correcting code G.** Recovering the code would be sufficient for an attacker to compromise the whole system. However it is placed between the permutation and the scrambling, both acting as protections. Moreover, it is not clear in which form it will be stored in each distinct implementations: while some might choose to store the Goppa polynomial and its associated elements, keeping the constructor matrix is also possible as it is equivalent. Additionally, if a fault is carelessly injected around this operation, it might be corrected as if it was a voluntary error, thus canceling any effect potentially helping attackers.

- **The permutation matrix P.** It is a matrix representing a transformation simply permuting the bits composing the input. Evidently, it has to be invertible. On the encryption side, it is located almost at the end of process, right before the adding of errors, while on the decryption one, it is at the very beginning, which makes it relatively accessible. As its associated operation is occurring before the error correction, here too an attacker can see his injected faults be rectified as if nothing irregular happened, hence the difficulty to exploit this area. However, disclosing the permutation matrix is, on its own, enough to invalidate the whole key. Indeed, at this point, one can assimilate a Goppa code to an alternant code in order to find G thanks to the public key. And S and G can be kept together, because SG and G produce the same code [PRD+15].

Hence, the selected target will be the permutation matrix, both for its potential in giving access to the secret key, and for its location, thereby, its accessibility for an offensive actor.

3.2.6 The permutation matrix in the McEliece cryptosystem

When depermuting the encrypted data block at the beginning of the decryption process, one is restructuring the underlying code, which allows to use a fast decoding algorithm. One who does not have P separately is prevented from using a similar one because he knows that (1) the bits in the data block are not in the correct order, and (2) some of them are toggled. The permutation matrix is thus paramount in this cryptosystem.

Figure 3.3 shows an example of permutation matrix. They follow specific constraints. It has to be a square matrix, as it also has to be invertible: the bits should be able to get shuffled to their original position. When looking at such a matrix, there should only be one bit set per row, and one bit set per column. In other words, each bit set in the matrix should only have zeros above it, under it, to its left and to its right. This is because each set bit will, during the associated transformation, be in charge of moving one bit of the entry to a new position. If a vector-matrix multiplication with the vector on the left, as illustrated in Figure 3.4, is considered, then the height of a set bit in the permutation matrix selects the position of the corresponding bit in the input vector, and its horizontal location selects its position in the output vector.

Another important property of permutation matrices is that they retain the Hamming weight of a vector when applied to it: the number of set bits in the output should be strictly the same as in the input vector. This is expected from an operation only moving bits without altering them, and the McEliece cryptosystem would not be able to work without it: it would mean that

$$\begin{bmatrix}
0&0&0&\mathbf{1}&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&\mathbf{1}&0&0&0&0\\
0&\mathbf{1}&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&\mathbf{1}&0&0&0&0&0\\
0&0&0&0&0&0&0&0&\mathbf{1}&0&0&0\\
0&0&0&0&0&0&0&0&0&0&\mathbf{1}&0\\
0&0&0&0&0&\mathbf{1}&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&\mathbf{1}&0&0\\
\mathbf{1}&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&\mathbf{1}\\
0&0&0&0&\mathbf{1}&0&0&0&0&0&0&0\\
0&0&\mathbf{1}&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

Figure 3.3: Example of a permutation matrix, of size 12×12.

$$\begin{bmatrix} 1&0&0&0&1&1&0&1&0&1&0&1 \end{bmatrix} \times \begin{bmatrix}
0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1\\
1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0
\end{bmatrix} = \begin{bmatrix} 0&0&1&0&1&0&1&1&0&0&1&1 \end{bmatrix}$$

Figure 3.4: Example of a vector-matrix multiplication, with a permutation matrix of size 12×12.

when voluntarily adding errors during encryption, one would not know the actual amount of corruption it would match to, before permutation.

During decryption, if one could inspect the intermediate variable right after the depermutation, before the decoding, it would be trivial to recover P: by sending a vector with a Hamming weight of exactly one, and seeing where the set bit ends up in this variable, repeatedly. However, as most implementations operate in a black box model or are obfuscated, this is not possible, or is quite difficult. Modifying the permutation matrix itself is a tempting possibility. But getting information on what happened at this stage by looking at the global output of the decryption is complex, as the decoding process and the unscrambling greatly affect the whole data block. Also, by modifying the properties of the permutation matrix, many expectations from the decoding algorithms will not be met. For example, if the permutation matrix does not preserve the Hamming weight anymore, the decoding process will likely often run into corrupted results. Thus, controlling the effects of a fault injected directly in P is quite complicated.

A classic, common way of implementing vector-matrix multiplications in $\mathbb{F}_2$ is by creating an accumulator and looping over each element of the vector. For each of them, if it is set, the corresponding line in the matrix is XORed to the accumulator. If it is not, then nothing is done. This method is widespread because it is simple, easy to implement and to debug. Also, it does not require manual bit-wise operations like masking for example: the full registers can be XORed together. As the vector and the matrix will likely be larger than the native length of the registers, each line will require more than one variable to be stored. For instance, on a 16-bit processor, if one wants to store a matrix of size 48×48, he will have to use three full variables for each line. In this general case, the multiplication process will imply two nested loops: one

for looping over each element of the vector, as before, and another one for XORing each variable of the corresponding line with the accumulator, if needed. This process is illustrated in details in Figure 3.5.

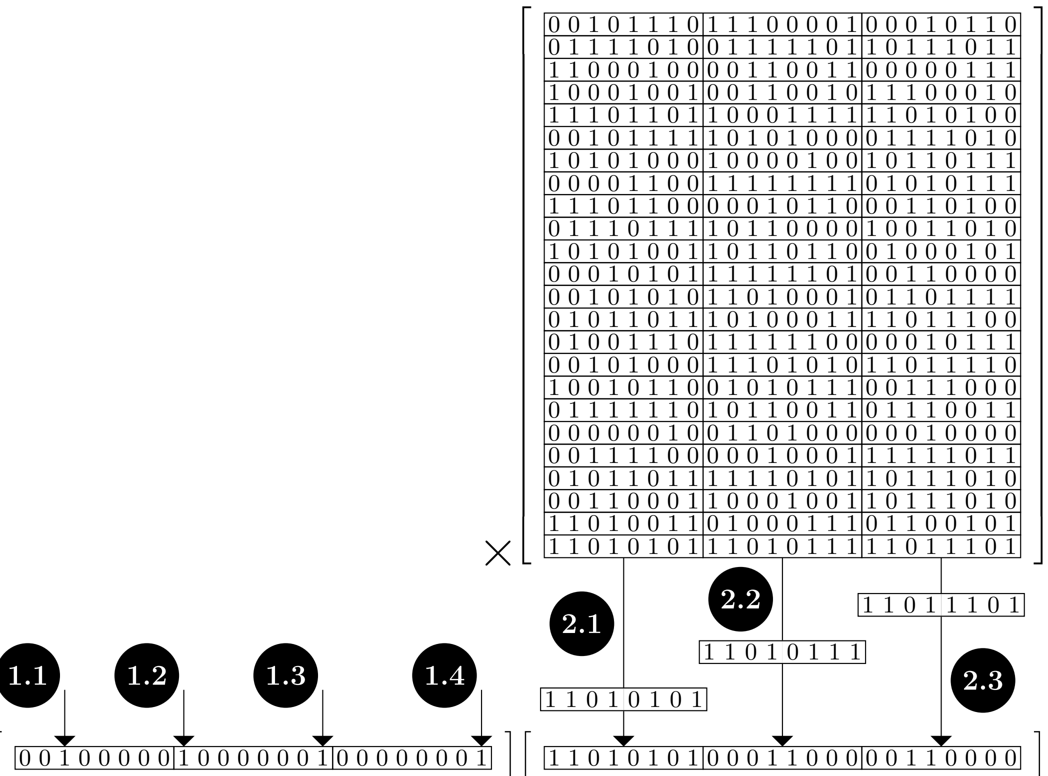


Figure 3.5: Illustration explaining the most common way of multiplying in $\mathbb{F}_2$ a large vector and a large matrix, stored using respectively an array of variables, and an array of arrays of variables. Here, the vector has a length of 24 bits, and it is multiplied with a 24×24 matrix, but everything is happening on a processor with 8-bit registers. Two loops are involved. The first one, whose iterations are represented by the **1.x** *labels, read the vector operand bit by bit, and trigger the second loop if and only if the read bit is set to one. The second loop, whose iterations are represented by the* **2.x** *labels, apply an exclusive or of the corresponding line's variables to the accumulator's variables, one by one.*

If such a process would happened to be implemented using the C programming language, which is very common in embedded devices, it would certainly be similar to Listing 3.1. It is important to note the use of the ˆ symbol, which the compiler will translate as a XOR between the two registers mentioned as operands.

3.2.7 Exploiting the Arm instruction set

When doing vector-matrix multiplications in $\mathbb{F}_2$, the use of a XOR instruction is not negotiable. Indeed, in Galois fields, the absence of carry makes it impossible to use a classic, conventional adding instruction. In Arm-based processors specifically, this precisely implies the exploitation of the Exclusive OR (EOR) instruction. A single, atomic operation like this one cannot be obfuscated: it needs to be interpreted by the processor, and thus have to respect the proper formatting. Figure 3.6 presents the typical format of an instruction in this family.

Just like data, code segments get loaded to the main memory and can be cached. If a fault would happened to be injected into code during this operation of loading, it would be present in it during use by the processor, and would exist for multiple executions. Inspecting instruction

Listing 3.1: Common vector-matrix multiplication code in C for 32-bit registers. The vector contains 1024 bits, and is multiplied with a 1024×1024 matrix. Each element of the `vector` *array contains 32 bits of it, and the* `matrix` *double array contains lines which store bits by group of 32, just like* `vector`*. The* `accu` *array is the accumulator, using the same storing method.*

```
uint32_t accu[1024/32] = {0};
for(int i = 0; i < 1024; i++)
{
  if(((vector[i/32] >> (31-(i%32))) & 0x01) != 0)
  {
    for(int j = 0; j < (1024/32); j++)
    {
      accu[j] = accu[j] ^ matrix[i*(1024/32)+j];
    }
  }
}
```

31 28	27 26	25	24 21	20	19 16	15 12	11 0
Cond	00	I	OpCode	S	Rn	Rd	Operand 2

Figure 3.6: Representation of 32-bit instruction designed for Arm processors.

corruption thus become technically sound. As the format for 32-bit Arm instructions indicates, the operation code is stored in 4 bits. Table 3.1 lists all of them.

The operation code for EOR is 0001. By making only one bit change in this, one can turn it to 0011, which is Reverse SuBstract (RSB). This operation is a subtraction on integers, outside of $\mathbb{F}_2$. One can consider replacing the central EOR of vector-matrix multiplication corresponding to the permutation during decryption in the McEliece cryptosystem by a RSB. This will have major implications: the Hamming weight conservation property will not be ensured anymore. The output of the permutation might not have the same Hamming weight as the input. In order to see if an attacker could take advantage of this, further inspections are required.

3.2.8 Information extraction tactic

When functioning properly and when given proper inputs, the decryption in the McEliece cryptosystem evidently returns a correct data block: the same one as before the encryption. If its inner workings were to be corrupted, or if it was fed improper data, two possibilities may occur:

1. the error-correcting code would fail to work, and the algorithm would encounter an error, leading either to a corrupted output, a crash, or an unexpected stop
2. the intermediate variable between the depermuting and the correcting code could be corrupted to the point where the nearest neighbor is another data than the one expected. In this case, the decryption algorithm would apparently work fine, but would return a different data block than the initial one, from before encryption

In fact, these behaviors represent a chance of getting information about what is happening around the permutation. Another property exhibited by the McEliece cryptosystem and linear error-correcting codes will be used: **a block full of zeros is necessarily a valid codeword**. Indeed, in the encryption process, if one multiply a block with no set bit inside by the *SGP* matrix, as expected, then the result will inevitably be another block with no set bit inside. In fact, if it was not the case, the considered correcting codes would not be linear. After this step, errors are voluntarily added. And it becomes possible to infer another property of the

Mnemonic	Code	Description
AND	0000	Rd = Rn AND Operand 2
EOR	0001	Rd = Rn EOR Operand 2
SUB	0010	Rd = Rn - Operand 2
RSB	0011	Rd = Operand 2 - Rn
ADD	0100	Rd = Rn + Operand 2
ADC	0101	Rd = Rn + Operand 2 + C
SBC	0110	Rd = Rn - Operand 2 + C
RSC	0111	Rd = Operand 2 - Rn + C
TST	1000	set condition codes on Rn AND Operand 2
TEQ	1001	set condition codes on Rn EOR Operand 2
CMP	1010	set condition codes on Rn - Operand 2
CMN	1011	set condition codes on Rn + Operand 2
ORR	1100	Rd = Rn OR Operand 2
MOV	1101	Rd = Operand 2
BIC	1110	Rd = Rn AND NOT Operand 2
MVN	1111	Rd = NOT Operand 2

Table 3.1: List of available operation codes in the 32-bit mode of Arm processors.

McEliece cryptosystem: **if an encrypted block has a Hamming weight inferior to t, then it corresponds to a clear block full of zeros**. Indeed, if a ciphered block has t or less ones in it, then the nearest neighbor is undoubtedly, necessarily an empty block, regardless of the permutation (the positions of the errors do not matter as long as the whole data stays close enough from all zeros) and regardless of the scrambling (bringing diffusion to zeros will result in zeros). In the end, this provides a strong tool to attackers: a large set of defined encrypted blocks can be associated with one specific, expected decryption output, which is an empty block. The stakes here are to exploit this tool with the help of fault injection to gain information of the permutation matrix.

In order to have any hope of mastering this knowledge, it is required to understand how to manipulate Hamming weights using subtractions. In particular, an efficient way of exercising manipulation like this is by using operands with only one bit set in them. Figure 3.7 shows some examples of such subtractions.

It can be noted that:

- when the first operand is greater than the second, the result's Hamming weight is proportional to the distance between the two set bits, as exposed in Figure 3.7a, Figure 3.7b, and Figure 3.7c.

- when the second operand is greater than the first, the result's Hamming weight is almost proportional to the distance between the Most Significant Bit (MSB) and the set bit in the second operand, as exposed in Figure 3.7d and Figure 3.7e.

On the other hand, but still on the subject of Hamming weights, the behavior of subtractions with only one bit set in any of the two operands should also be studied. Figure 3.8 contains examples of them.

Here, it can be seen in Figure 3.8a that when the bit set to one is in the first operand, the result will be coherently equal to it, thus implying the same Hamming weight as it. However, when the only bit set is in the second operand, the result's Hamming weight is proportional to the distance between the MSB and the one, as shown in Figure 3.8b.

Incidentally, a specific point requiring attention is the order of the subtraction an attacker will create. In the Arm architecture (see the instructions' format in Figure 3.6), subtractions can

```
  0x 0 0 4 0 0 0 0 0
- 0x 0 0 0 0 0 2 0 0
  0x 0 0 4 0 0 0 0 0
+ 0x F F F F F E 0 0
  0x 0 0 3 F F E 0 0
```

(a)

```
  0x 0 0 0 1 0 0 0 0
- 0x 0 0 0 0 4 0 0 0
  0x 0 0 0 1 0 0 0 0
+ 0x F F F F C 0 0 0
  0x 0 0 0 0 C 0 0 0
```

(b)

```
  0x 0 2 0 0 0 0 0 0
- 0x 0 0 0 0 0 0 0 8
  0x 0 2 0 0 0 0 0 0
+ 0x F F F F F F F 8
  0x 0 1 F F F F F 8
```

(c)

```
  0x 0 0 0 0 0 2 0 0
- 0x 0 0 4 0 0 0 0 0
  0x 0 0 0 0 0 2 0 0
+ 0x F F C 0 0 0 0 0
  0x F F C 0 0 2 0 0
```

(d)

```
  0x 0 0 0 0 0 0 8 0
- 0x 0 0 0 0 0 1 0 0
  0x 0 0 0 0 0 0 8 0
+ 0x F F F F F F 0 0
  0x F F F F F F 8 0
```

(e)

Figure 3.7: Examples of subtractions where both operands contain only one bit set. All values are in hexadecimal form.

```
  0x 0 0 0 1 0 0 0 0
- 0x 0 0 0 0 0 0 0 0
  0x 0 0 0 1 0 0 0 0
+ 0x 0 0 0 0 0 0 0 0
  0x 0 0 0 1 0 0 0 0
```

(a)

```
  0x 0 0 0 0 0 0 0 0
- 0x 0 0 0 1 0 0 0 0
  0x 0 0 0 0 0 0 0 0
+ 0x F F F F 0 0 0 0
  0x F F F F 0 0 0 0
```

(b)

Figure 3.8: Examples of subtractions with only one bit set in both of the operands. All values are in hexadecimal form.

be implemented with the SUB instruction, which subtracts the value of the second operand to the first one in the Rn section. It can also be implemented using the RSB instruction, which also does a subtraction, but in the inverse order: the value of the first operand in Rn is subtracted to the second operand. This is useful since the second operand brings many more possibilities to refer to the value, its address, or the register holding it.

However, if a XOR operation like the one in Listing 3.1 is considered, there is no guarantee that the order of the generated subtraction will be the same as this one, since it does not matter when doing exclusive ors ($a \oplus b = b \oplus a$). Thus, if one injects a fault to transform a EOR into a RSB, it is not certain if the operation, originally XORing a line of the permutation matrix with the accumulator vector, will end up subtracting the matrix line from the accumulator, or subtracting the accumulator from the matrix line. From the point of view of an attacker, determining an implementation's subtraction's order is possible, and is even necessary as the attack's methodology will change depending on it.

Let p be the size of a platform's registers in bits. In the case of Arm ones, it is often 32. Let h be the Hamming weight on the crafted inputs. Here is how an attacker can get information about the permutation matrix, after transforming the EOR instruction into a RSB one, by only controlling the input and observing the decrypted output.

- **If the assembly code subtracts the matrix lines from the accumulator** ($accumulator = accumulator - matrix\ line$), then he should:

  1. craft a random input vector with a Hamming weight $h = \left\lceil \frac{t}{p} \right\rceil$, or $h = \frac{t}{p} + 1$ if $\frac{t}{p}$ is

already an integer.

2. process it through the faulted decryption implementation.
3. analyze the result:
   - if the execution went properly and returned a data block full of zeros, then the error correction was not overloaded. The attacker knows this because, as a reminder, if there is less than t bits in the input vector, this is the only correct decryption possible. He cannot infer anything about the permutation matrix from this result.
   - if the execution crashed, returned an error, or apparently went properly but returned a result which is not a data block full of zeros, then it means the attacker overloaded the error correction. **He can acknowledge that the permutation matrix's lines associated to the ones in the input vector all have their own one in different groups of p columns**. This is the case because each time the subtraction between the accumulator vector and a matrix line occurs, the group of p bits in the accumulator where there is a one in the matrix line will get an unknown number of ones stored in it. Only zeros will be subtracted from the other groups of p bits in the accumulator which will thus not change. In consequence, the only way to overload the error correction here is if each ones in the subtracted matrix lines are in different, distinct groups of p bits. This condition, while being necessary, is not sufficient: the attacker cannot know the number of ones present in each group of the accumulator at the end of the faulted depermuting.
4. repeat steps 1 to 3 to gain more and more information about the permutation matrix.
   - if t is near a multiple of p while being below it, causing error overloads might be harder than usual. Once it becomes clear only some positions in the input vector provoke them, the attacker can increment the Hamming weight he selected at step 1, and only use the positions that never caused error overloads at this point. The Hamming weight can be incremented again for the positions that still do not provoke corruptions, and so on.
   - if t is near a multiple of p while being above it, causing error overloads should be frequent from the start. At some point, positions that did not trigger any corruptions, and only them, can be used with a Hamming weight $h = \left\lceil \frac{t}{t - \left(p \times \left\lfloor \frac{t}{p} \right\rfloor\right)} \right\rceil$. Then $\left\lceil \frac{t}{t - \left(p \times \left\lfloor \frac{t}{p} \right\rfloor\right) - 1} \right\rceil$ can be used for the remaining positions only, and so on.

   At some point, the attacker can group or separate the permutation matrix's lines depending on if they have their one in the same group of p bits or not. Additionally, for each line, the difficulty to trigger error overloads with it gives an indication about where the one is in it: as exposed earlier, the more the one is near a register's MSB, the less ones will be induced in the result stored in the accumulator, indicating that if it is difficult to overload with a specific line, then its one is likely close to the MSB of the variables containing it.

- **If the assembly code subtracts the accumulator from the matrix lines** ($accumulator = matrix\ line - accumulator$), then he should:

1. craft a random input vector with a Hamming weight $h = \left\lceil \frac{t}{p} \right\rceil + 1$.
2. process it through the faulted decryption implementation.
3. analyze the result:
   - if the execution went properly and returned a data block full of zeros, then the error correction was not overloaded. The attacker knows this because, as a reminder, if there is less than t bits in the input vector, this is the only correct decryption possible. He cannot infer anything about the permutation matrix from this result.
   - if the execution crashed, returned an error, or apparently went properly but returned a result which is not a data block full of zeros, then it means the attacker overloaded the error correction. **He can acknowledge that the permutation matrix's lines associated to the ones in the input vector all have their own one in different groups of p columns**. This is the case because the first time there is a subtraction of the accumulator from a matrix line, the latter will get copied in the accumulator, since the former is empty. And from now on, if the ones in the selected matrix lines are in different groups of p bits, for each new subtraction, the group corresponding to where there is a one in the matrix line will get copied in the accumulator, and the accumulator's groups already having a single or multiple ones in it will end up having an unknown number of ones in them. In consequence, the only way to overload the error correction here is if each ones in the selected matrix lines are in different, distinct groups of p bits. This condition, while being necessary, is not sufficient: the attacker cannot know the number of ones present in each group of the accumulator at the end of the faulted depermuting.
4. repeat the previous steps to gain more and more information about the permutation matrix, until the attacker can group its lines depending on if they have their one in the same group of p bits or not.

3.2.9 Operands order explanation and determination

As the above methodology emphasizes, the order of the subtraction is important. It is quite unpredictable, since it is influenced by two different erratic parameters. The first one is the programmer itself. Since the order of the operands does not matter when XORing them, he might write the operation in one way or the other in the source code. The second parameter is the compiler. For the same reason, it does not provide any guarantee on the conservation of the operands' order, as it is useless on an unfaulted assembly code. In fact, the inversion of the operands by the compiler is a recurrent event. Listing 3.2, which is generated from the code in Listing 3.1, shows an example.

Listing 3.2: Extract from a built version of vulnerable code from Listing 3.1 into ARM-assembly, around the XOR operation.

```
@ |                         Instructions                    |
@ | Address | Binary value |        Mnemonic                |
     10660       e51b300c       ldr r3, [fp, #-12]
     10664       e1a03103       lsl r3, r3, #2
     10668       e24b2004       sub r2, fp, #4
     1066c       e0823003       add r3, r2, r3
     10670       e5131024       ldr r1, [r3, #-36]
     10674       e51b2008       ldr r2, [fp, #-8]
     10678       e1a03002       mov r3, r2
     1067c       e1a03083       lsl r3, r3, #1
     10680       e0832002       add r2, r3, r2
```

```
10684     e51b300c     ldr r3, [fp, #-12]
10688     e0822003     add r2, r2, r3
1068c     e59f30d8     ldr r3, [pc, #216]
10690     e08f3003     add r3, pc, r3
10694     e7933102     ldr r3, [r3, r2, lsl #2]
10698     e0212003     eor r2, r3, r1
1069c     e51b300c     ldr r3, [fp, #-12]
106a0     e1a03103     lsl r3, r3, #2
106a4     e24b1004     sub r1, fp, #4
106a8     e0813003     add r3, r1, r3
106ac     e5032024     str r2, [r3, #-36]
```

In this assembly code, one can spot the targeted EOR instruction at address `0x10698`. By looking at how are prepared both of its parameters, one can deduce that `r1` refers to `accu[j]`, and `r3` refers to `matrix[i*(1024/32)+j]`. This assembly instruction is thus responsible for providing `matrix[i*(1024/32)+j] ^ accu[j]`, while the source code, on the other hand, described `accu[j] ^ matrix[i*(1024/32)+j]`. While the inversion caused by the compiler does not matter as is, it is relevant once the instruction is faulted. Here, when the EOR is replaced by a RSB in this example, the result is `accu[j] - matrix[i*(1024/32)+j]`, because it is a reversed subtraction.

In any way, from the point of view of an attacker, the order of the subtraction must be determined. For this, he can use the effects of the value of h. Indeed, one possible method is to, first, find an input vector with a Hamming weight $h = \left\lceil \frac{t}{p} \right\rceil + 1$ generating an overload of the error correction, and second, see what is happening when removing one bit set in it, so that his Hamming weight becomes $h = \left\lceil \frac{t}{p} \right\rceil$. If it is still generating an overload of the error correction, then the implementation operates subtractions in the $accumulator = accumulator - matrix\ line$ order. Otherwise, if it does not, then the order is $accumulator = matrix\ line - accumulator$. This method of determination works because the attack requires less bits set in the input vector when the order is $accumulator = accumulator - matrix\ line$, thus when $h = \left\lceil \frac{t}{p} \right\rceil + 1$ achieves corruption but $h = \left\lceil \frac{t}{p} \right\rceil$ does not, this is necessarily because the additional bit set was required, implying more ones are needed to overload the error correction, as it is the case with $accumulator = matrix\ line - accumulator$.

3.2.10 Practical example when $accumulator = accumulator - matrix\ line$

This section will describe in details the conduct of the attack when, after injecting the fault, the order of the subtraction is $accumulator = accumulator - matrix\ line$. For the sake of clarity and layout preservation, the chosen parameters will be largely inferior to usual, real ones. The following explanation will use $n = 9$ and $t = 4$. Note that in a real world scenario, these parameters would actually be incorrect. This is because in general, one needs to have $mt + 1 \leq n \leq 2^m$ [Ber11]. On the left side, this implies $4m + 1 \leq 9 \implies m \leq 2$. But on the right side, this implies $9 \leq 2^m \implies log_2(9) \leq m$. In a concrete error-correcting code, this would not work because m cannot be smaller than 2 and bigger than approximately 3.17 at the same time. However, in the framework of this explanation, this will not be an issue.

For the good execution of this example, the ease of reading and mostly to keep it concise, a fictional processor with registers of size $p = 3$ will be considered. No real Arm implementation actually exhibits such registers. Finally, for the same reason of brevity, optimal input vectors will be crafted: an actual attacker would have to experience more trials and errors before gaining the same bits of information.

The following random, secret permutation matrix will be considered:

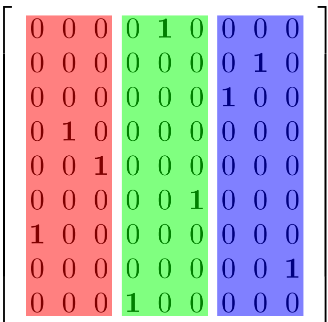


Figure 3.9: A random permutation matrix.

As expected it is a 9×9 matrix. As $p = 3$, the groups of 3 columns are highlighted in red, green and blue. The goal is not to determine into which group the one of each row is going into, it is rather to which rows have their one in the same group. Once the implementation is faulted as desired, an attacker can start using input vectors with $h = \left\lceil \frac{t}{p} \right\rceil$ as Hamming weights, and the invalid multiplications occuring as depermuting to cause overloads in the error correction.

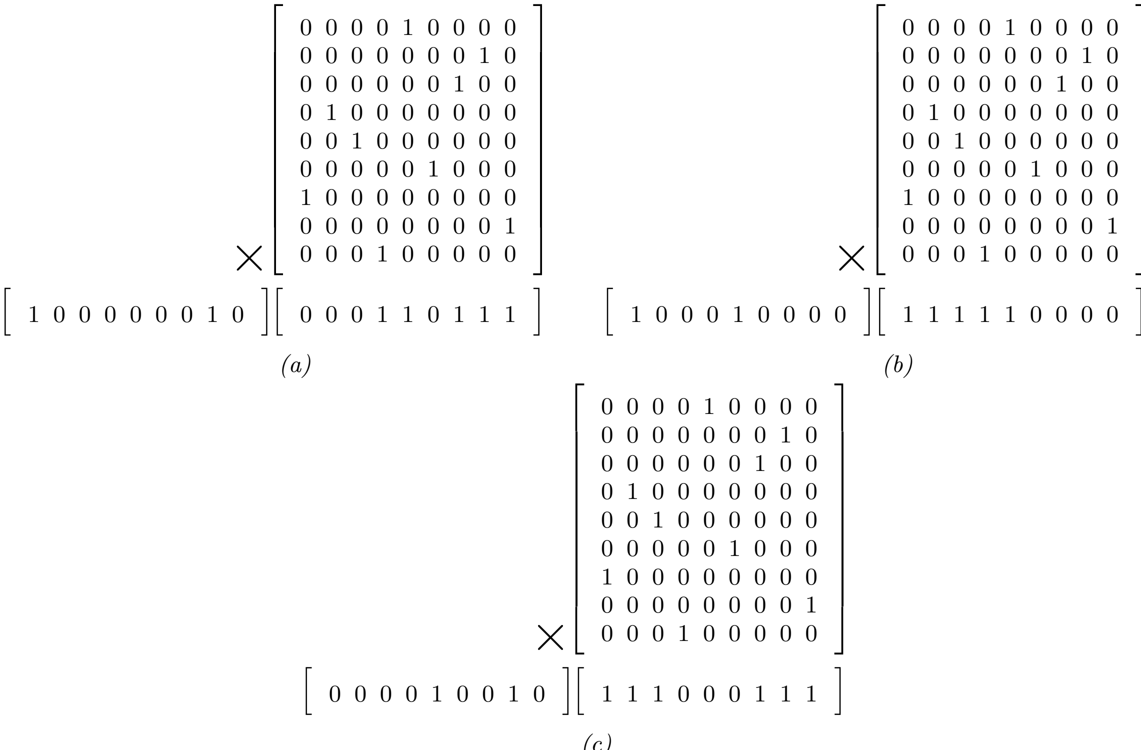


Figure 3.10: Three examples of faulted multiplications giving information about the permutation matrix.

The three faulted multiplications illustrated above cause an overload, since the results have Hamming weights above t, which is equal to 4. In order of appearance, they indicate the attacker that:

- lines 1 and 8 of the permutation matrix have their respective one in different groups of p columns
- lines 1 and 5 of the permutation matrix have their respective one in different groups of p columns

- lines 5 and 8 of the permutation matrix have their respective one in different groups of p columns

Again, this is the case because it is the only way it could happen. These three bits of information are precious they tell an offensive actor that lines 1, 5 and 8 each a their one in a different group of p columns. However it is impossible to associate groups to lines. An attacker would continue sending similar inputs:

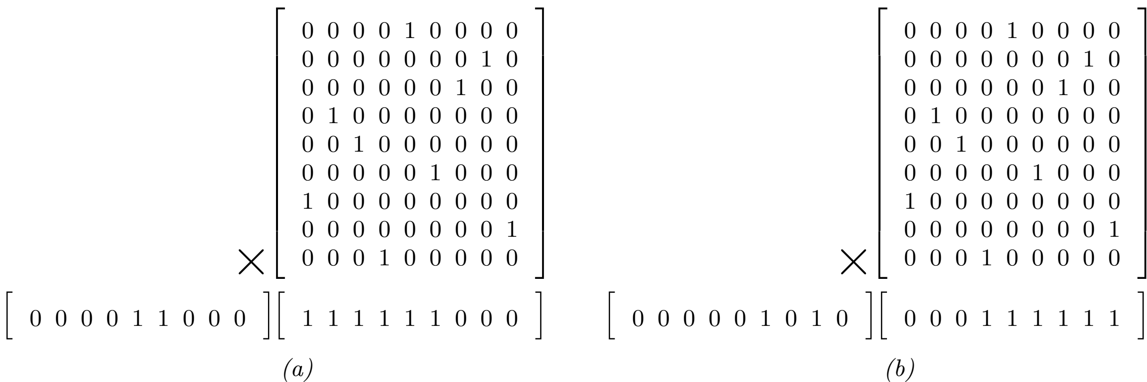


Figure 3.11: Two examples of faulted multiplications giving information about the permutation matrix.

The two flawed multiplications above tell an attacker that the sixth line of the permutation matrix has its one neither in the same group as the fifth one or the eighth one. Then, the sixth line can be associated with the first line: it necessarily has its one in the same group of p columns, no other classification is possible.

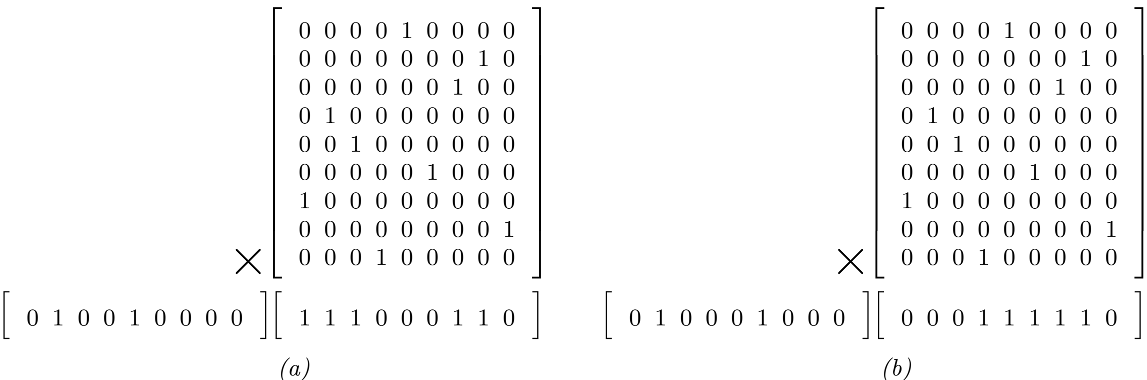


Figure 3.12: Two examples of faulted multiplications giving information about the permutation matrix.

The two error overloads caused by the flawed operations above indicate to an attacker that the second line has its one in the same group as the eighth line. Indeed, the corruptions would not be possible if its one was in the same group as the fifth or sixth line.

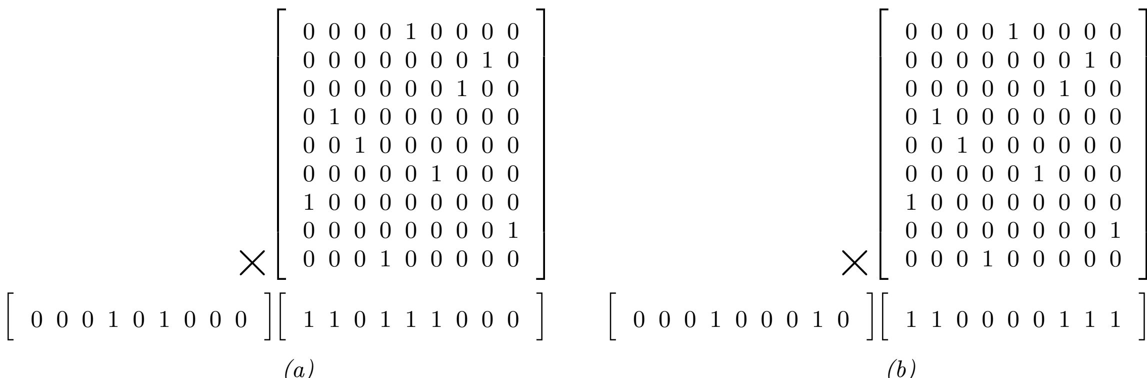


Figure 3.13: Two examples of faulted multiplications giving information about the permutation matrix.

The two flawed multiplications above, both resulting with a Hamming weight strictly superior to t, separate the fourth line from the sixth and the eighth respectively. Then, the fourth line of the permutation matrix necessarily has its one in the same group as the fifth one. Additionally, an attacker can consider the fact that he was not able to cause any corruption at this point with positions 3, 7 and 9. Finally, from all this data, he can confirm that:

- lines 1 and 6 have their one in the same group of p columns
- lines 4 and 5 have their one in the same group of p columns
- lines 2 and 8 have their one in the same group of p columns
- lines 3, 7, and 9 have their one in different group of p columns

The attacker can thus represent the grouping of lines in this way for lines 1, 2, 4, 5, 6, 8:

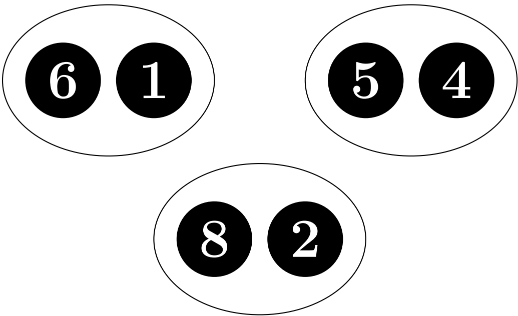


Figure 3.14: Representation of exclusive sets of lines that have been found.

And in this way for lines 3, 7 and 9:

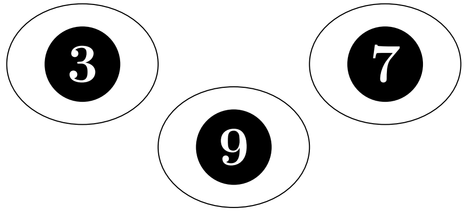


Figure 3.15: Representation of exclusive sets of lines that have been found.

Moreover, specifically in this example, the impossibility to trigger overloads with lines 3, 7 and 9 indicates that:

- lines 3, 7, and 9 have their one in the MSB of any group
- lines 1, 2, 4, 5, 6, and 8 have their one in any positions besides the MSBs

Now, in order to recover the remaining data about the permutation matrix, brute forcing is required, in order to search only the remaining space, by respecting all of the above rules and constraints.

3.2.11 Practical example when $accumulator = matrix\ line - accumulator$

This section will describe in details the conduct of the attack when, after injecting the fault, the order of the subtraction is $accumulator = matrix\ line - accumulator$. Here too, for the sake of clarity and layout preservation, the chosen parameters will be largely inferior to usual, real ones: $n = 9$ and $t = 4$. Again, in a real world scenario, these parameters would actually be incorrect for the same reason already given in subsection 3.2.10. In this example as well, it will however not be an issue.

With the same motives, about the registers' size, here too p will be set to 3, even if no Arm processor actually exhibits 3-bit registers. Finally, again, instead of describing the many trials and errors an attacker would have to go through, only optimal inputs will be detailed.

The following random, secret permutation matrix will be considered:

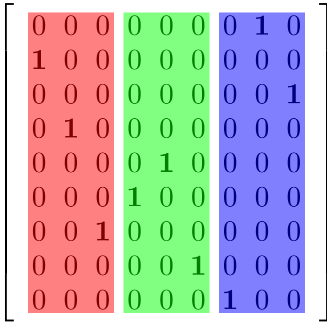


Figure 3.16: A random permutation matrix.

This is a 9×9 matrix as expected. As $p = 3$, the groups of 3 columns are highlighted in red, green and blue. Once again, the goal is not to determine into which group the one of each row is going into, it is rather to which rows have their one in the same group. Once the implementation is faulted as desired, an attacker can start using input vectors with $h = \left\lceil \frac{t}{p} \right\rceil + 1$ as Hamming weights, and the invalid multiplications occuring as depermuting to cause overloads in the error correction.

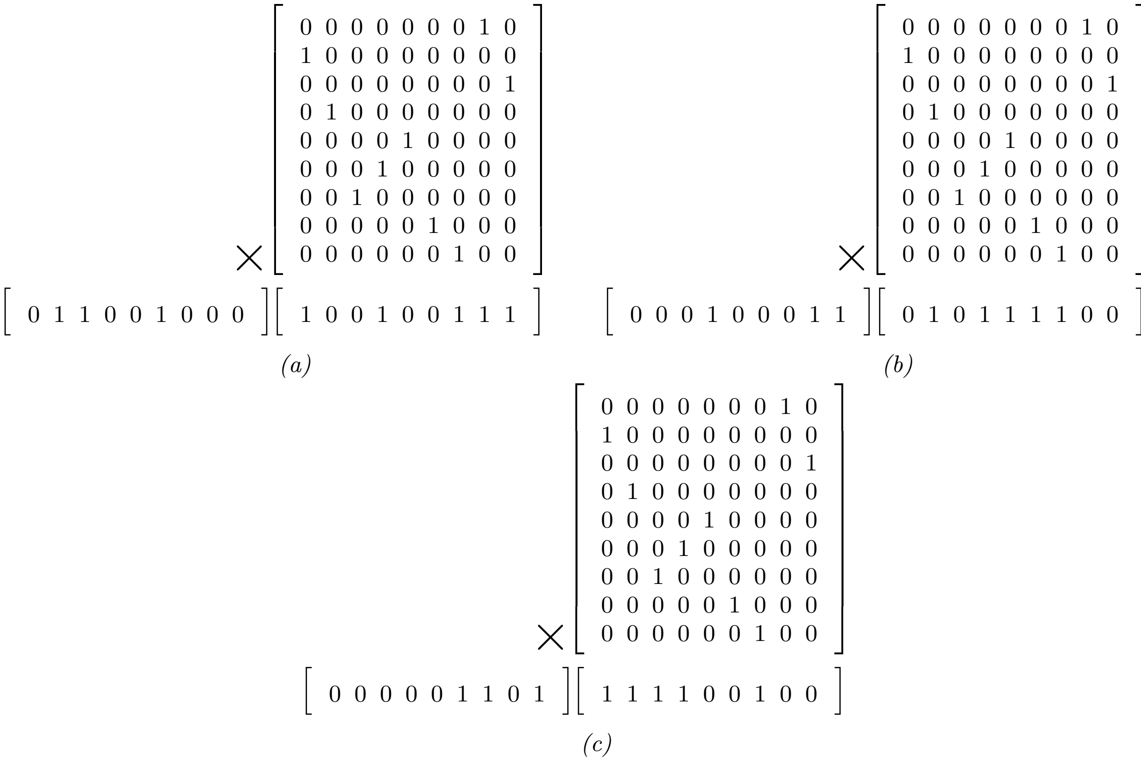


Figure 3.17: Three examples of faulted multiplications giving information about the permutation matrix.

Each of the three flawed multiplications above is useful. Indeed:

- the first one tells the attacker that lines 2, 3 and 6 have their respective one in different groups of p columns
- the first one tells the attacker that lines 4, 8 and 9 have their respective one in different groups of p columns
- the first one tells the attacker that lines 6, 7 and 9 have their respective one in different groups of p columns

He can confirm this as if this was not the case, the Hamming weight of the improper permutation's outputs could not be strictly above $t = 4$, and would not trigger error correction overloads. The attacker can continue sending crafted input to the implementation.

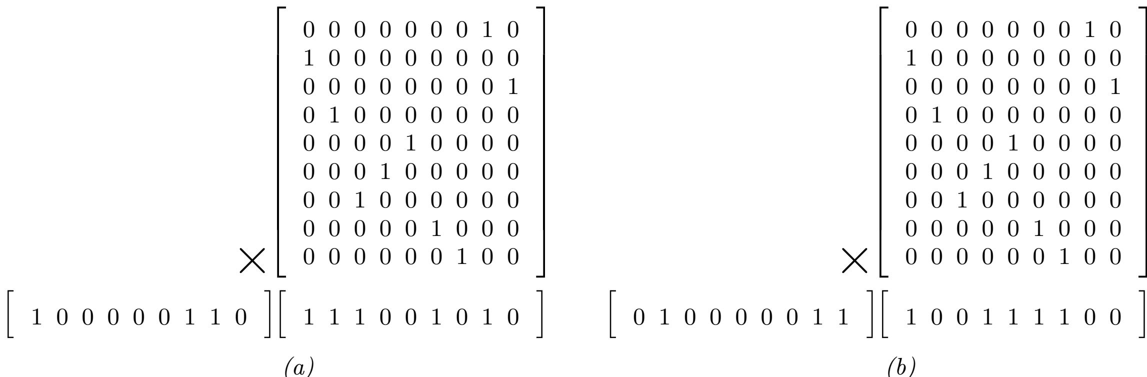


Figure 3.18: Two examples of faulted multiplications giving information about the permutation matrix.

The two failed multiplication above provoke overloads as well. The first one tells the attacker that lines 1, 7 and 8 have their respective one in different groups of p columns. The second, in

the same way, separates lines 2, 8 and 9. From now on, the attacker does not need to process more input vectors. By using simple logic, he can group the lines of the permutation matrix depending on if they have their one in the same group of p columns, like this:

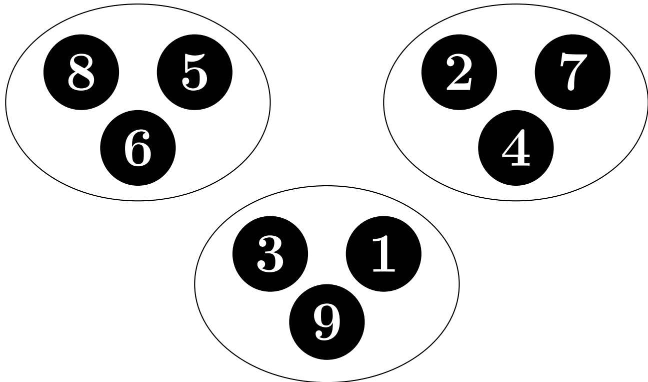


Figure 3.19: Representation of the exclusive sets of lines that have been found.

This is the only grouping respecting the lines' separations observed earlier. The fifth line was not witnessed in any overload, but the attacker does not need it to: the only remaining spot for it was with the eighth and sixth lines. Again, he does not know which group of lines is assigned to which group of columns (represented in red, green and blue previously). Brute forcing is needed to recover these last bits of information about the permutation matrix.

3.2.12 Resulting metrics

Describing the effects of this attack is difficult. This is the case because its methodology varies greatly depending on several factors: the parameters of the targeted McEliece cryptosystem implementation, the subtraction order at the assembly level... A good way to estimate a general result is to consider the drop in entropy on the permutation matrix for common values of p, when the attacker has classified every line of it. In practice, the data recovered about the permutation matrix will end up at this scale, even if it is a little less, or even more (as observed in subsection 3.2.10, where the attacker obtained some small indications regarding the closeness of some lines' ones to the variables' MSB). In this spirit, Figure 3.20 and Figure 3.21 represent the entropy loss when considering two iconic n values: 1024 and 6960. The first one is the size mentioned in the original specification of the cryptosystem by Robert J. McEliece himself [McE78]. The second one is the size advocated for by the PQCRYPTO's cryptographic experts and researchers when considering "long-term secure post-quantum systems" [DWTN15].

As a reminder, the original entropy of a permutation matrix of size $n \times n$ is not $log_2(2^{n^2})$, like any common matrix in $\mathbb{F}_2$, but less: $log_2(n!)$. It is logically based on the number of possible permutations of an n-bit vector's elements. But when an attacker has managed to fully partition the matrix, meaning he achieved the definition of smaller sets sorting all the lines gathered together or not depending on if they have their respective ones in the same group of p columns, the entropy becomes $log_2\left(p!^{\frac{n}{p}} \times \frac{n}{p}!\right)$, when $\frac{n}{p}$ is an integer. $p!$ is the number of possible combinations inside each group. $\frac{n}{p}$ is its power because there are this number of groups. Even if the elements inside them are sorted, the position of groups of columns themselves need to be searched, and there are $\frac{n}{p}!$ of them.

The fact that the post-attack entropy become lower with smaller values of p is expected. Indeed, due to its combinatorial nature here, the entropy has a strong exponential character. An attacker would thus prefer to brute force a large number of small sets, rather than a small number of large sets. On this subject, it is important to remember that such an attacker would

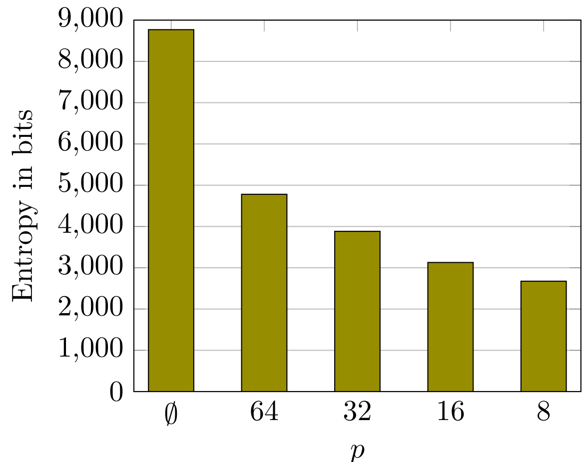


Figure 3.20: Comparison of the permutation matrix's entropy before and after the attack, with multiple p values, for $n = 1024$. The ∅ value refers to the entropy prior to the attack.

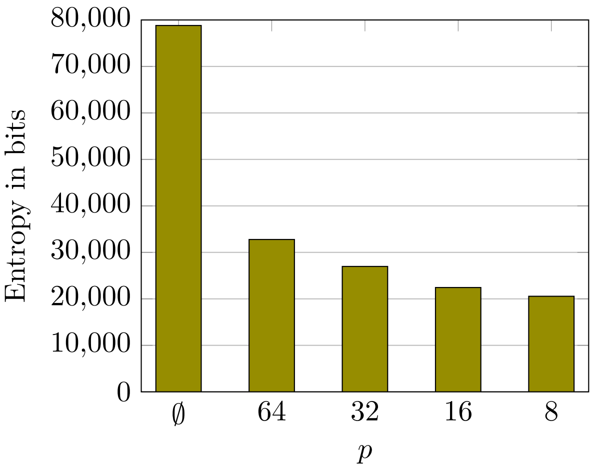


Figure 3.21: Comparison of the permutation matrix's entropy before and after the attack, with multiple p values, for $n = 6960$. The ∅ value refers to the entropy prior to the attack.

know all the parameters: p should be the targeted processor's registers' size, n is the codewords' length, k is the length of clear blocks of data, and t should be indicated by the owner of the private key, otherwise the person in charge of the encryption could add to many errors when processing a block of data.

3.3 White-box measures

3.3.1 Context and motivations

One could consider the previous attack only in the framework of hardware faults, applied with dedicated tools. This would be an error. While it is a valid, worthy of interest scenario, there is nevertheless another one that can be quite challenging for developers and solutions providers. It is the one of a implementation dedicated to running on uncontrolled environments. Since they consider a very powerful attack model, these will resort to protections like obfuscation before, during or after compilation, in order to prevent offensive actors to recover such an executable and reverse-engineer it, allowing a key recovery or other unwanted consequences.

The obfuscation engines found in the industry always employ several different methods to

make binaries hard to understand. They rely on techniques including, but not limited to, modification of the call graph, alteration of the control flow, data masking, and substitution of instructions. This last one is particularly relevant in the framework of the attack presented here, as it relies heavily on the EOR instruction commonly used in classic vector-matrix multiplications in $\mathbb{F}_2$. However, the replacement of instructions is often based on mathematical properties available in $\mathbb{R}$ or in $\mathbb{Z}$, but not in Galois fields. In result, programs making use of EOR to make additions in $\mathbb{F}_2$ will still have this instruction somewhere in them. In this case, in such an open context, an attacker can dump the executable of an implementation, scan for EOR codes in it, and try the methods presented here on them. This possibility directly impacts the viability of the McEliece cryptosystem in open, uncontrolled environments.

In addition, if one considers the application of this attack only via software, meaning using instrumentation or debugging tools, it will likely be more efficient than the results exposed in subsection 3.2.12. This is because instead of being limited to a one-bit change in the operation code, restricting thus the possibilities to changing the EOR into a RSB, the only which can provide the desired aspects, the attacker can now change to any operation code present in Table 3.1. Then, one can infer that the SUB, ADD, ADC, SBC, RSC, ORR and MVN instructions are susceptible to be good heuristics for undertaking similar attacks, some certainly giving better results than with RSB. This is undoubtedly the case since some of them can increase the Hamming weight of the result much more easily, thus it can be expected that triggering error correction overloads with them is more straightforward, or even that it is possible to, maybe, manage to get more information about the positions of the ones in the permutation matrix in different ways. Others, however, like AND, TST, TEQ, CMP, CMN, MOV and BIC, do not give much hope in this perspective.

In this context, the option of a white-box implementation of the McEliece cryptosystem, based on more traditional methods, like precalculation of intermediate variables, can be quite attractive. It would go beyond the simple obfuscation methods, and provide a major advantage: since all values would be precomputed, the EOR instruction, targeted by all of the offensive work presented here, would simply not exist in the executable. An attacker thus could not replace it with anything. As always with white-box implementations, encodings and bijections should be put in place to prevent an easy reading of all the data, which would expunge this benefit.

Another argument makes the McEliece cryptosystem rather appealing when considering white-box implementations; it is the relatively ease with which precomputation can be considered on it. Asymmetrical algorithms based on factorization (like RSA) or on lattices (like Kyber [Nat23]) operate deeply with large intermediate values that can hardly be subdivided, or cut into portions so to speak, which remove any hope of precalculation, since the implied number of results to operate and store would be inconceivable. In the McEliece cryptosystem however, many tactics can be put in place to achieve acceptable sizes, as the next sections will explain.

3.3.2 Decryption

Naturally, any white-box implementation of the McEliece cryptosystem will focus on protecting the private key, during the decryption. Securing the public key, which is already accessible by anybody, is not a priority. The original decryption method in the McEliece cryptosystem is represented in Figure 3.2. It is based on two matrix multiplications, and a decoding.

One might be tempted to simply merge each of the depermuting and unscrambling matrices with a random linear application (meaning with a random invertible matrix of the same size), that could be cancelled outside of the algorithm. However, this method introduces risks towards attacks based on interpolation. To properly protect these assets, the techniques used cannot rely exclusively on linear fusions.

The matrix multiplications can seem impossible to precompute, since one takes a n-bit input,

and the other takes a k-bit input. Processing 2^k and 2^n results is inconceivable. However, one can apply here a matrix decomposition technique already used on the `MixColumns` step for white-box implementations of AES [CEJP03b]. Indeed, it is possible to subdivide the large matrix into smaller ones, accessible to precalculation, as illustrated in Figure 3.22. The size of the smaller matrices is entirely up to the designer; it is also possible to choose one that is not a divider of the total matrix size.

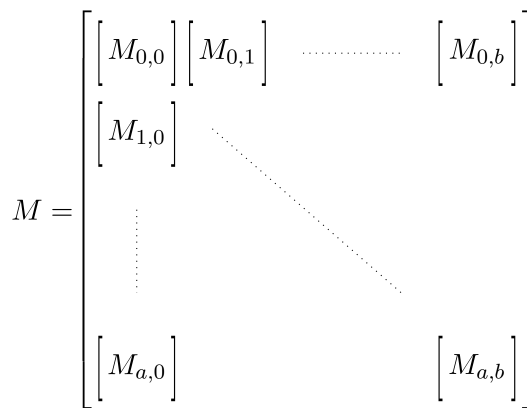


Figure 3.22: Visualization illustrating the decomposition of a matrix M in order to make the precomputation of its associated transformation possible. a is the number of submatrices rows, and b is the number of submatrices columns.

Once such a decomposition has been decided, precalculation will only be required on each of the submatrices. As a matter of fact, the total result of a multiplication with the total matrix can be obtained by consecutively adding up the precomputed multiplications with the submatrices. If u is a vector multiplied with M on its left, then the result of $u \times M$ is composed of b concatenated subvectors:

$$u \times M = \left[\sum_{i=0}^{a} u_i \times M_{i,0} \parallel \sum_{i=0}^{a} u_i \times M_{i,1} \parallel \sum_{i=0}^{a} u_i \times M_{i,2} \parallel \ldots \parallel \sum_{i=0}^{a} u_i \times M_{i,b} \right] \tag{3.1}$$

The addition of the intermediary results themselves should also be precomputed, this way the entire linear application would only consists of tables in the end. Here, since the algorithm works in $\mathbb{F}_2$, the additions would actually be XOR operations. In the context of decryption, this technique of matrix subdivision can potentially be applied to both the depermutation matrix P^{-1} and the unscrambling matrix S^{-1}. However, the only remaining step, the decoding process, cannot be trivially precomputed because of its input size, most likely above 1024 bits. An adaptation work is required for it.

One possible approach consists in using multiple, small, correcting codes, instead of a single large one. Each subcode would actually be associated with its own subpermutation (which would thus not need matrix subdivision), however they would all share a common unscrambling matrix that would mix the data across all the subcodes. This strategy is represented in Figure 3.23.

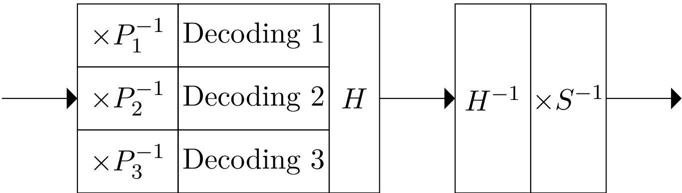


Figure 3.23: Representation of the operational principles of an explored McEliece cryptosystem modification, applied to decryption. In this example, the number of subcodes is set to 3.

This way, if the sizes are chosen carefully, the subcodes should be precomputable. To protect the private key during the deciphering step, an internal confusion layer, based on random bijections and linear transformations, here named H, can be applied at the end of the subcodes, with its inverse at the beginning of the common unscrambling application. Note that the subpermutations and subcodes could be merged and precomputed together, but the unscrambling matrix should be subdivided and precomputed alone.

One crucial aspect to consider in this approach is that the depermutation matrices, which are critical to the security of the McEliece cryptosystem, may have less entropy than the large equivalent one. This is due to the fast growth of the factorial function that describes their complexity based on their size. Therefore, the size of the permutation matrices must be chosen carefully, taking into account both security and usability constraints. To ensure the robustness of this idea, many offensive approaches still need to be undertaken.

3.3.3 Encryption

Even if modifying the encryption is originally not necessary, as it only operates on the public key, it now becomes required because of how the propositions modify the decryption. The original encryption method in the McEliece cryptosystem is represented in Figure 3.1. In order to make it compatible with the previously explained approach, the large code must be, here too, replaced by smaller subcodes. These subcodes would each be associated with their own subpermutation of the same size. At the end of the multiplication, errors should still be added to the result, but considering each section specifically, as each of them will have its own maximal number of bit flips it can accept. The scrambling matrix, here as well, remains unique and common to all the subcodes. These aspects are illustrated in Figure 3.24.

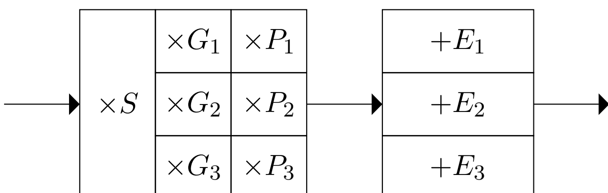


Figure 3.24: Representation of the operational principles of an explored McEliece cryptosystem modification, applied to encryption. In this example, the number of subcodes is set to 3.

These modifications of the encryption are almost invisible to the end user. Indeed, the subcodes and the subpermutations can be efficiently embedded inside the large, single SGP matrix usually used in the original McEliece cryptosystem. Only the voluntary adding of errors must be slightly adapted, if it was for it, the encryption process would not have changed at all.

Synthesis

In this chapter, the context leading to a new interest in the McEliece cryptosystem is recalled. The way it works is described, along with its advantages and disadvantages. Its recognitions by different actors and institutions are detailed.

Then, a more offensive view is taken, and the history of attacks against the McEliece cryptosystem is given. In particular, it is recalled how, while several side-channel-based attacks were discovered against it, none based on fault-injection has been achieved against its private key. Thus, the presentation of one that has been undertaken focusing on Arm implementations follows. A precise description of how it works and how it targets a common way of exploiting the permutation matrix is given, along with some resulting metrics.

In the end, this offensive is put back in the context of uncontrolled environments, since this type of attack can help to circumvent obfuscation. As a countermeasure, some white-

box measures that could be applied to the McEliece cryptosystem are given, which would help produce implementations immune against this fault injection.

Chapter

Inter-process confidentiality in Android

Abstract

The management of energy in portable devices is a complex task: ensuring both safety and long lifetimes requires lots of expertise and work on both the hardware and the software. The expectations of the end users, too, are elevated: they also desire a fast charge time, an estimation of the remaining charge and lifetime to the percent and to the minute respectively, and batteries staying healthy for many years. Moreover, mobile systems almost always rely on lithium batteries. These are common because they have an excellent power density, and many other advantages. However, they are quite unpredictable, hard to master, and can reveal themselves to be dangerous if use improperly. In consequence, device constructors now often use fuel gauges. These are Integrated Circuits (ICs) dedicated to monitor the battery and the power of the platform, thus taking in charge major responsibilities in dealing with energy.

After having described this context, this chapter details how these components are implemented in Android smartphones and tablets, first from the hardware point of view. Then, how they are embedded on the software side, and how the OS integrates them. The commonly accepted rules in Android about the different interactions possible are recalled, specifically about the accesses between applications, and accesses from one application to resources in the underlying layers, potentially from hardware. The security policy Android deploys specifically concerning fuel gauges is studied and explained. Thereafter, in regards to it, several plausible risks are given: spying of the end user, having two applications secretly communicating, and having one malicious application spying on a legitimate one.

Since it might have serious consequences, a focus is undertaken specifically on the spying of PIN code entries by the user, on a legitimate application. It is shown how, in Android, the lenient security policy about fuel gauges helps in achieving such offensive maneuvers. This attack is described with its two main parts: the acquisition of data, and the exploitation of the data. Since, as a result, it can allow to get a small set of possible PIN sequences (often below 10), several aspects of this offensive are discussed, and possible countermeasures at different abstraction layers are given.

Scientific valorization and recognition

This work led to acceptances from the following events, journals or institutes:

- *Symposium sur la Sécurité des Technologies de l'Information et des Communications* (SSTIC) 2023, for: *Batterie à bord, quand les jauges de carburant dépassent les limites*
- *International Workshop on Security* (IWSEC) 2023, for: *Power analysis pushed too far — Breaking Android-based isolation with fuel gauges.* Associated publication in *Lecture Notes in Computer Science* (LNCS) volume 14128
- Patent application number WO2024121142 filed at the *Institut National de la Propriété Industrielle* (INPI) on December 6, 2022: *Procédé de protection d'un terminal contre une*

attaque par canal auxiliaire

Chapter content

4.1 Power management in embedded devices . . . 56
4.1.1 Power saving and its stakes . . . 56
4.1.2 The challenges of managing rechargeable batteries . . . 57
4.1.3 Better measurements for better decisions: the fuel gauges . . . 59
4.2 Risk assessment . . . 60
4.2.1 Fuel gauges hardware integration . . . 60
4.2.2 Android's architecture and abstraction stack . . . 62
4.2.3 Android's policy around fuel gauges . . . 64
4.2.4 Identified risks . . . 66
4.3 Sensitive data recovery through fuel gauges . . . 67
4.3.1 Testing tools and experimental conditions . . . 67
4.3.2 Data recovery . . . 69
4.3.3 Data exploitation . . . 70
4.3.4 Discussion about the results . . . 73
4.3.5 Countermeasures . . . 75

Services providers developing solutions targeting the white-box security model often do so because they will be deployed on COTS products. When considering payments, transports and identification, smartphones and tablets are the machines of choice since they are mobile and easy to carry. Since implementations can try to rely on the isolations provided by their environment, it is paramount to evaluate how sound this is. Thus, investigating on how applications interact between them and with the hardware resources is relevant. In particular, power management is an interesting aspect to look into: it must necessarily exist in a mobile device, it is hard, it can have severe consequences if implemented improperly, and applicative contents can sometimes have a say in it.

4.1 Power management in embedded devices

4.1.1 Power saving and its stakes

The management of power on mobile devices is paramount and implies many things. Usually, the first goal is to get the longest battery life possible. On a theoretical level, there are only two ways of achieving this:

- Increase the available energy on the device, i.e. implementing larger batteries
- Reducing the consumption of the platform, so that the embedded energy is, actually and literally, saved

Manufacturers have heavily used the first method. Modern smartphones usually embed batteries with capacities around 2500 and 6000 mAh, and tablets are frequently in the 6000 and 11000 mAh range. However, this way of increasing run time is mostly constrained by hard physics limits. The energy density of the batteries cannot be raised indefinitely. And the size of mobile machines, in order to remain ergonomic, need to stay restricted. Batteries are necessarily quite heavy, too. Also, while increasing the capacity works to have a bigger operating life, it also extends the charging delay: having a bigger power tank allows to stay unplugged longer, but this naturally requires to stay plugged longer too, to fully recover the potential energy.

The second method, which is about reducing the consumption, has many advantages. First, it does not imply a longer charge time, as it does not influence the capacity of the battery. It is considered more environmental friendly, as it preserves energy and is not essentially based on the exploitation of more resource. Incidentally, it can also be applied to non-mobile systems, which benefit from a constant, powerful source of energy. Finally, power saving also lead to less heat generated by the Joule effect. However, reducing the consumption of devices requires efforts in different and complex tasks:

- **Optimizing the hardware**. In processors and calculators in general, this implies lithography processes with thinner cells, since they require smaller power draws. As of today, chips with a 3 nm lithography can be found in the market, but going further than this is truly difficult, because the manufacturing processes are bound by physics constraints. Power optimization via hardware can also affect several other components, like vibrators, speakers, flashlights, or cameras. Screens, on smartphones, are roughly responsible for almost half of the energy consumption. This is the case because they emit a lot of light, but here too, their physical optimization is complex, and they already are quite efficient.
- **Optimizing the software**. This basically relies on, first, avoiding unnecessary operations as much as possible. These can include things like context switching, background activities, and heavy animations with GPU work, for instances. Secondly, it also relies on making necessary operations consume the smallest amount of energy. For example, when waiting for a resource, passive interruptions should be preferred to active polling. SIMD and parallel operations can be preferred to single, one-at-a-time processes. Finally, more generally, on Android, it is interesting to note that there is no consensus on the definition of what is necessary and what is not. Third party developers trying to get their applications to run in background or in sleep mode frequently get their processes killed by the OS, often for arbitrary reasons, and regularly despite the desires of the users [Nal22]. This topic is a subject of dispute between, on one side, Google along with devices' constructors, and on the other, third party developers.

In the end, if getting the longest battery life is almost always cited as the number one priority, with reducing the charge time as secondary, there is another one, widely forgotten, and non-negotiable: safety. Indeed, lithium can be a dangerous component if used improperly. Knowing the remaining charge and remaining lifetime is also a common desire from end users.

4.1.2 The challenges of managing rechargeable batteries

Since the dusk of disposable batteries and the dawn of rechargeable ones, lithium has been the obvious choice for all mobile devices, for several decades. Batteries based on it offer a very high energy density along with low self-discharge. They do not suffer from memory effect. They are quite affordable, especially due to the worldwide scale of their production today. When produced, stored, and used properly, they are considered fairly safe. Compared to other technologies, they also exhibit good lifetimes, from creation to being considered unsatisfactory by the users.

On the other end, controlling lithium-based batteries, and predicting as well as analyzing their behavior is a challenging task. While capacitors have, ideally, a tension directly proportional to their charge, this cannot be expected at all about these batteries. In fact, even when considering a perfect model of one of them, this relation is far from being linear. Figure 4.1 shows the typical behavior of such batteries' Open Circuit Voltage (OCV) versus their state of charge. The OCV is the voltage at their poles when not connected to anything. The state of charge is a ratio referring to the available capacity compared to the one when fully charged.

Lithium batteries' voltage rapidly change when they are near to being fully charged or fully depleted. On the contrary, at the middle of their state of charge, they have what is often called a *flat zone*, where the voltage change very little during the depletion. In it, basing any

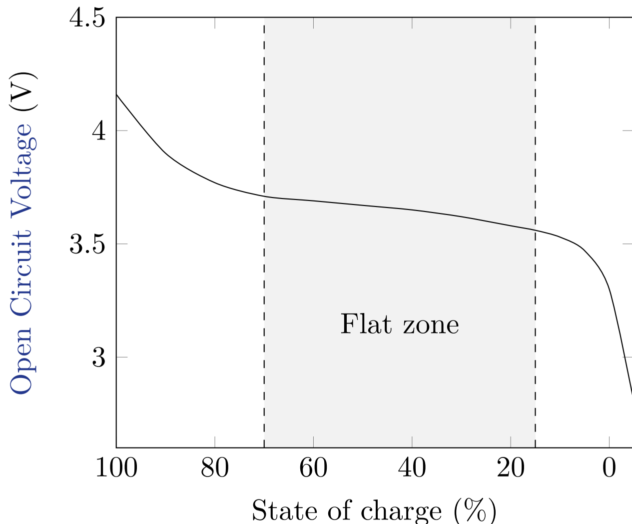


Figure 4.1: The typical OCV of a lithium battery depending on its state of charge. The x-axis is reversed to represent the discharging of a battery, going from being full (100%) to being empty (0%). The curve goes beyond the 0% since, by convention, it is defined to be the point where the system should crash because the tension is too low; nevertheless, the battery still presents some voltage and capacity after such an event [RBSC17].

estimation on the voltage is hard, since the order of magnitude is intensely small. Moreover, the measurement in itself is not a trivial matter, and imply all the classic issues with Analog to Digital Converters (ADCs): the sampling rate and the resolution must be carefully chosen, dealing with the calibration and the noise is required, and some drift can also be present. Above all, the behavior of the batteries differ greatly depending on the operating temperature, on the age of the battery, on their chemicals along with their quality of build, and on how they are used. Thus, providing a modeling of a battery is complex and costly: in practice, it imply spending a lot of time in a laboratory, studying the behavior of one when its numerous parameters are influenced.

The bad management of lithium batteries can also lead to danger. This chemical has an elevated explosive power, often triggered by high temperatures. It is the most important source of laptops, smartphones and tablets getting on fire. One of the countermeasures for this is to use a separator inside the battery, between the anode and the cathode, which is supposed to purposefully melt at a temperature above the regular, operating ones, but under the dangerous ones: usually around 135°C and 165°C. This cuts the flow of energy, i.e. stops the functioning of the battery. But improper manufacturing of it can make it hazardous.

Another difficulty with lithium batteries comes up when they are composed of multiple cells. Depending on the system's design, these can operate at different temperatures one from the other. A delta above 10°C is quite common. These are undesirable, as they will induce wear at different speeds on each cell, which will thus behave differently as time goes on, further complicating management, if taking into account such aspects is actually possible on a specific device.

Charging is frequently underestimated, as the notion of lifetime does not have any value in this context; however it is not a trivial operation [TF16]. Charging time need to be considered and reduced as much as possible. The charging voltage must be carefully chosen, sometimes requiring a communication with the charger. Overcharging a battery and failing to precharge one that is completely empty will inevitably harm it, whether it is in a immediate way or on the long term.

On a platform made for one specific function causing a constant energy consumption, the

remaining lifetime of a battery can more or less be associated to its state of charge. However, in the context of smartphones, this is very much not the case. It is impossible to know the future use of it by its user, and more importantly, during communications, the electrical consumption is a function of distance between the device and the nearest cellular base station, which is also impossible to estimate ahead of time. Remaining lifetime thus have to deal with an additional load of uncertainty.

4.1.3 Better measurements for better decisions: the fuel gauges

When designing portable devices, tackling all these issues is thus extremely complex. During development, it requires a lot of knowledge, skill, and time. Therefore, it is quite costly. The numerous techniques that one can put in place to help in this task are sophisticated.

In order to estimate the state of charge, two ways can be considered. The first is watching the voltage. However it must be done when the battery is rested, meaning not being charged, and not heavily powering the platform. This technique is also hardly applicable in the flat zone of the OCV curve, because of the very small scale of voltage change in it. The second way to make an estimation is by integrating the output current. It necessitates a Coulomb counter, based on a resistor, which helps measuring it.

Whatever the signal used to make the measurements, getting a state of charge from it will require a theoretical model of the battery, which will act as a function giving the desired results depending on all the influencing parameters, including but not limited to, the temperature and the battery's age. Accurate modeling of a battery is a complex task, but when knowing the remaining charge is necessary, the calculation can then simply be done at run time. Another way of getting the state of charge from the measurements is by using a multi-dimensional table assigning such estimations for many values of voltage, current, age and temperature. No calculation is needed here, but the table containing all the numbers will naturally be considerably large, and thus imply a strong memory requirement. Moreover, for the batteries' makers, the compilation of all these values will likely involve months of experimentation. And in the end, all these efforts might not really pay off: many factors can impact the precision provided by table lookups, the main one being the impedance variations originating from the manufacture, that can be up to a few percents between cells [WL21].

Many different solutions are usually put in place to ensure safety when operating lithium batteries. They focus on avoiding voltages and currents too high or too low, inappropriate temperatures, short circuits, or failures with the fuse and during communications. For example, charging a battery can be quite dangerous, and imply various precautions: outside of the 10 and 45°C range, it must be done with lower voltages and currents, and must be completely stopped when largely below or above it [Jap21]. Even when applying these rules, the potential delta of temperature between cells still must be taken into account. In the end, to comply with specifications, extend battery life, and reduce charging time, the best option is to be able to communicate with the charger, to specify the preferred voltage and current at any given time. Also, keeping a log of the occurring data is useful for post mortem investigations.

An important part of a proper management of batteries is to learn their behavior on the long term, as the device is being used for weeks, months, or years. By constantly updating the data about them, measurements and estimations can become a lot more precise. In practice, this imply things like actualizing the limits of the battery every time it gets almost empty, i.e. when the user gets to approximately 5% left. This adjustment is precious but many users rarely get to this point with their devices, and some sensitive machines just cannot afford to. Continuously tracking the impedance of the battery is a more available alternative, but the theory behind it is more complicated, and it requires a more active monitoring.

In the face of all these features that are challenging to develop, constructors often have recourse to components called *fuel gauges*, or *gas gauges*. Contrary to what these names suggest,

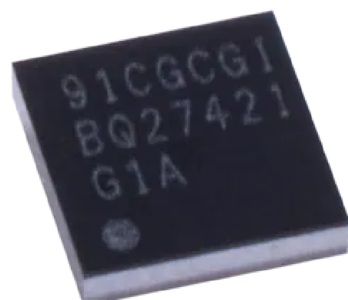
they are fully dedicated to observing and analyzing the behavior and metrics of electrical batteries. Their use is comparable to a delegation of responsibility: the designers will not have to deal with the power sources, and will only have to interact with the fuel gauge in order to get information about them. In practice, a SoC can ask them for measurements like the battery's voltage, its temperature, the current going in or out of it, and for estimations like the absolute remaining charge, the state of charge, and the remaining lifetime. They will try to track the age of the battery, in order to refine their calculations and results. They can also manage pluggings and unpluggings for the platform, and take some responsibilities during the charging. Finally, when they are embedded in the batteries' package, they can contain some cryptographic primitives, in order to provide authentication: the most common use of this feature is to prevent the devices from booting if the battery they contain is not authentic, or not from an authorized source.

By gathering such a large set of considerations and features, one can expect fuel gauges to give precise measurements and estimations about batteries. And in reality, this has been verified: the values they return indeed are impressively accurate, to the point that on a smartphone, it seems like some specific events are detectable on signals related to power [BXZ+17]. This instantly raises some questions on the subject of security: can this be a threat to confidentiality on the platform? There should not be any risk related to hardware, since fuel gauges only monitor power and do not control it like a Power Management Integrated Circuit (PMIC). But on the software side, numerous uncertainties arise from this reality. In the face of these considerations, a full risk analysis becomes appropriate to determine the actual offensive possibilities that are unlocked here. Since they accumulate many sensitive responsibilities and are extremely widespread amongst the population, the focus here will be on machines running the Android OS.

4.2 Risk assessment

4.2.1 Fuel gauges hardware integration

Most of the time, fuel gauge ICs have their own dedicated package. They often have a square shape, with each side being only a few millimeters. Chip-Scale Packaging (CSP) and Wafer-Level Packaging (WLP) are common forms they can have, and a size smaller than 3×3 millimeters is usual. Figure 4.2 contains a two-sided view of a sample.



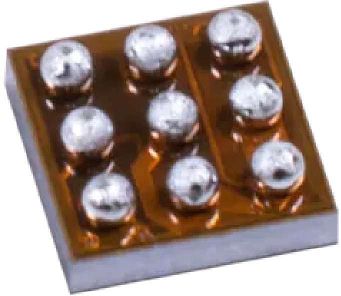


Figure 4.2: Photograph of a fuel gauge. This model is the bq27421-G1, commercialized by Texas Instruments.

They are directly soldered on the device's PCB, preferably close to the power sources: the battery and the energy plug. Their inclusion must thus be thought of when starting to design the platform and its various components, and their integration must be planned during manufacturing. Figure 4.3 illustrates this with a photograph representing how the Nintendo Switch embeds a fuel gauge.

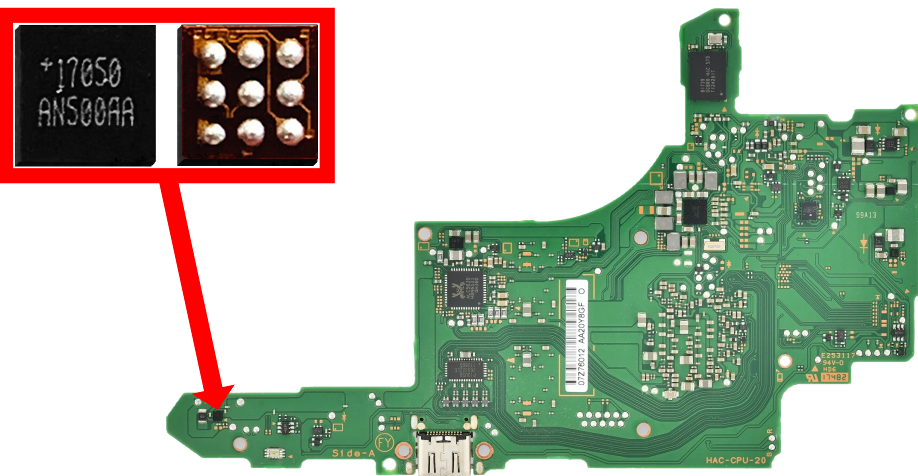


Figure 4.3: Photograph of a Nintendo Switch's motherboard [iFi]. It embeds a MAX17050 from Maxim Integrated, which is highlighted with a red arrow. A zoomed view, surrounded in red too, is provided.

On the connection level, fuel gauges should be located between the central system and the power sources. This position is expected as they monitor the battery along with the external source, and then report on them to the main environment, usually a SoC. Overall, the point is to have all energy, whether it is internal or external, go through the fuel gauge, so that it can watch it and do its measurements. Indeed, in order to follow the used current for example, it is necessary to have it passing through a Coulomb counter. All components can however have a common ground. Obviously, a serial link is required between the central system and the fuel gauge, so that they can interact. This communication channel typically corresponds to an I^2C bus, likely managed by a driver residing in the kernel space of the rich environment. Figure 4.4 contains a diagram showing the most usual fuel gauge hardware integration.

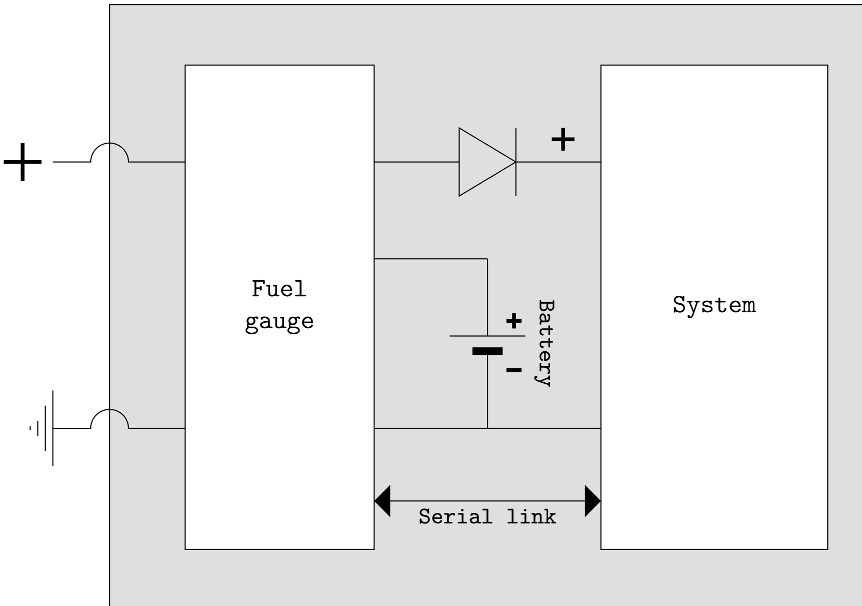


Figure 4.4: Diagram representing how a fuel gauge is typically integrated in a device. In practice, the system is generally a SoC.

If one has a device and wants to know if it embeds a fuel gauge, two options exist. The first one is to open it and visually check the PCB. These ICs are quite small but some reference should be written on them. Looking near the Universal Serial Bus (USB) plug gives the best result

if it is used as the power input. On Android phones and tablets, the second method to detect the presence of a fuel gauge in a device is based on software: one can probe all the equipment related to power in a terminal, using the shell command `ls -a /sys/class/power_supply`. Running this does not require having root access on the platform, and can thus be done by a regular user. A typical output is given in Listing 4.1 and can contain numerous elements. Here, amongst the different results, one is particularly eye-catching: `maxfg`. Indeed, with experience, it becomes obvious that `fg` is an acronym for *fuel gauge*, and `max` refers to a popular designer and manufacturer of these, *Maxim Integrated*. This detection of fuel gauges through shell commands is poorly documented, but is theoretically supported by all machines running Linux or Android. On an indicative basis, it can be noted smartphones and tablets from Google's Nexus and Pixel lines often embed these ICs.

Listing 4.1: Terminal output when determining the power-related equipments in the Pixel 6 smartphone. The `maxfg` name indicates the presence of a fuel gauge, namely the `MAX17050` in this specific case.

```
$ ls -a /sys/class/power_supply
battery
dc
gcpm
gcpm_pps
main-charger
maxfg
pca9468-mains
tcpm-source-psy-i2c-max77759tcpc
usb
wireless
```

Finally, an important note about fuel gauges is that they are quite expensive. A unit's price will almost always be above 1 euro. From an industrial point of view, this represents a large impact on Bills Of Materials (BOMs) and thus on potential revenues. In consequence, these ICs have more chance of being included in high-end, expensive products, than on low-end ones.

4.2.2 Android's architecture and abstraction stack

In the secure design of systems, determining how applicative contents can access hardware is crucial. In Apple products, the lower layers often are undocumented and inaccessible to the point of many features being impossible to implement, at least for third parties. The Android OS offers more possibilities in this regard. Also, since the devices running it are designed by many distinct constructors, the variability regarding these subjects is likely to be greater, hence the value in investigating around them.

The way the Android OS technically deals with applications and their access to hardware resources is quite homogeneous for all of them. All of third party executables have the same abstraction layer under them. However, what is allowed and what is not is more or less clear depending on which type of interaction is considered. Figure 4.5 provides a schematic representation different from Figure 2.3, more focused on a layered view. The bottom layer contains the Linux kernel and the drivers, along with a part of the SELinux security framework. The native user space is essentially composed of libraries and standalone executables not intended for virtual machines, like Google's Bionic, which is their in-house standard C library, and other, varied tools. A layer dedicated to virtualization contains ART, the runtime environment for applications in Android, along with Zygote, the process model from which is duplicated each new application launching. Finally, the Java framework can be considered as the heart of Android, as this is where most of its features offered to applications are located, along with its design, policies, behavior logics.

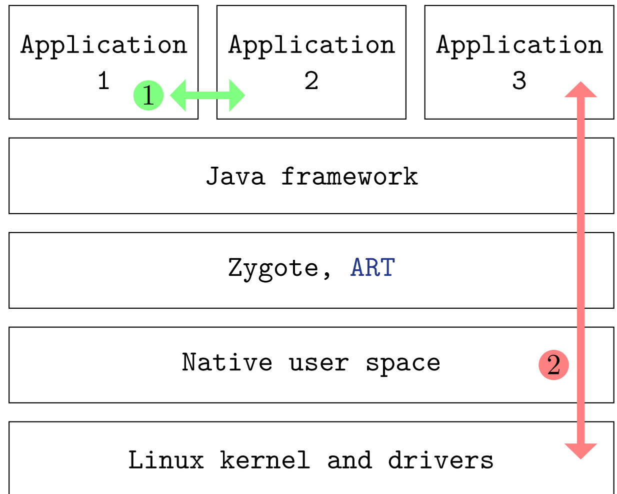


Figure 4.5: Simplified view of the abstraction model in Android systems. The first, green arrow represents horizontal accesses. The second, red arrow represents vertical accesses.

Interactions between applications, here called horizontal accesses, are represented in green in Figure 4.5. The rules for developers about them have been quite clear since the beginnings of Android: except via one specific way, they are simply forbidden and should not occur. If an application is able to get the data, settings, runtime information, files, or any content about another one, it is seen as a defect. The central exception to this relies in the concept of intents in Android. Applications onto this OS are composed of activities and services. The former can be summed up as screens or interfaces dedicated to a specific task. The latter allows to run time consuming tasks separately, principally in the background. Interactions between these are based on intents, meaning messages describing an action with its parameters. They can be:

- **Implicit intents.** They describe a general action that can be assigned to any available component on the system able to take charge of it.
- **Explicit intents.** They describe an action and the specific component expected to receive it.

Intents can also be used to broadcast messages, in order to inform about events on the platform. In practice, intents allow users to switch from one task screen to another, for instance when an application requires a file picker, and to trigger heavy operations in the background, like network accesses. An intent can be considered as a horizontal access since it allows one application to send data to another, through its embedded parameters. Overall, these mechanisms rely on Binder, the Inter-Process Communication (IPC) manager specific to Android.

Interactions between applications and the underlying resources, here called vertical accesses, are represented in red in Figure 4.5. They are the ones behind operations like having an application access the camera, read sensors values, communicate on the network, make the phone vibrate, and more. From an application's point of view, these accesses start through the Android framework API, and their requests are then relayed through many abstraction layers, up to their destination. Each of these layers technically has the power to stop or even modify the request. Their decision to do so or not leads to the main subject here: while horizontal accesses are dictated by explicit rules, this is less the case for vertical interactions. In Android, the decisions related to vertical interactions depend on the OS version, on the platform's constructor, on the nature of the solicited resource, and on the desired type of access: reading or writing. It is complex to keep track of all of these parameters. In result, for an application widely distributed on a large fleet of varied Android devices, trying to access hardware resources will likely imply to find

out at run time what is allowed and what is not. It becomes even more tricky considering the wide range of sensors now embedded in smartphones and tablets: light meters, accelerometers, gyroscopes, microphones, cameras, thermometers, etc... Indeed, they essentially each have their specific rules associated to them. Fuel gauges are no exception, and they have their very own dedicated access policy. Determining it is important.

4.2.3 Android's policy around fuel gauges

Since the early days of the Android OS, a system service called `BatteryManager` has been existing, and has been available to any application on the platform. At the beginning, it only allowed to recover the status of the battery regarding its health (`GOOD`, `OVERHEAT`, `DEAD`, `OVER_VOLTAGE`...) or its use (`CHARGING`, `DISCHARGING`, `FULL`...), as well as, if applicable, the charging source: `USB` (power comes from a USB port like the one of a computer, providing Direct Current (DC) electricity), or `AC` (power comes from an Alternating Current (AC) charger). At the time, the information provided by this service were quite scarce, and their different possible uses were thus quite limited.

The `BatteryManager` system service has then been expanded over time. In version 5.0 of Android (called *Lollipop*), constants were added to form queries that can potentially be redirected to a fuel gauge. These include:

- `CURRENT_NOW`: this value describes the measure of the instantaneous current in real time, entering or leaving the battery, in microamperes.
- `CAPACITY`: this value describes the remaining capacity inside the battery, in percentage.
- `ENERGY_COUNTER`: this value describes the remaining energy inside the battery, in nanowatt-hours.

At this point, this service became more interesting to developers targeting Android. Technically, any application running on this system can access the `BatteryManager` service, and request any of the available attributes, during its execution. This service is at the level of the Android Java framework, but can relay any demand to lower layers. If a measure request is deemed legitimate, it is actually necessary to do so in order to reach the fuel gauge. It should thus also go through the VM, and through the native user space, up to the kernel and its drivers, which can take in charge the hardware communication.

At this point, the obvious question that comes to mind is: does any of the layers under the applicative one in Android apply some kind of moderation concerning requests about power data? The kernel space does not. The presence of SELinux in it, deployed since version 4.3, makes it a candidate for enforcing some policy here, since it allows to implement completely arbitrary rules, however it does not. The drivers either, but this is more expected since it is not really their role. Another smart choice would be the Android Java framework layer: almost all of the time, and for most resources, this is where moderation happens in this OS. Nonetheless, no such thing can be found at this level concerning fuel gauges. Similarly, no moderation is present, neither at the ART level, nor in Zygote. Thus, during the early times of fuel gauges implementation in Android, accesses to their information were completely free to third party executable contents. To verify this assertion, tests have been executed on Android smartphones with software belonging to this era. It confirmed interactions with the fuel gauges are not moderated at all, whether it is in access frequency, or related to the type or nature of the accesses.

Another aspect that requires special attention is the ability to capture these measurements at any time, including when other applications are in use, or when the phone is in sleep mode. The `FOREGROUND_SERVICE` permission, introduced in Android 9, is required by the activity manager when an application demands to run a task in the background. Here too, this one is granted

Listing 4.2: Extract of Android's system sensor manager since version 12.

```
/**
 * Checks if a sensor should be capped according to
 * HIGH_SAMPLING_RATE_SENSORS permission.
 *
 * This needs to be kept in sync with the list defined on the native side
 * in frameworks/native/services/sensorservice/SensorService.cpp
 */
private boolean isSensorInCappedSet(int sensorType) {
    return (sensorType == Sensor.TYPE_ACCELEROMETER
            || sensorType == Sensor.TYPE_ACCELEROMETER_UNCALIBRATED
            || sensorType == Sensor.TYPE_GYROSCOPE
            || sensorType == Sensor.TYPE_GYROSCOPE_UNCALIBRATED
            || sensorType == Sensor.TYPE_MAGNETIC_FIELD
            || sensorType == Sensor.TYPE_MAGNETIC_FIELD_UNCALIBRATED);
    }
```

without any request from the user. However, to obtain this permission, a notification must be pushed to the list dedicated to this purpose in the system's graphical interface. Therefore, there are now many applications that require a permanent notification, so as not to be sacrificed by the battery saver, or to be able to receive communications directly without going through the Google services. This could create cases of spoofing, where an application that is supposed to be for chatting or playing a video game is actually probing the fuel gauge in the background.

The next consideration is how often the fuel gauge can be checked. From Android 12 onwards, the `HIGH_SAMPLING_RATE_SENSORS` permission has appeared in the Java framework. This permission is intended to limit the polling of sensors above 200 Hz. However, since it is a normal permission, it can be requested without visual warning to the user. Moreover, it does not concern fuel gauges in any case: this permission was implemented using a blacklist, as shown in the extract in Listing 4.2. However fuel gauges, for unknown reasons, were not included in it. In Android versions prior to 12, this measure does not exist at all.

In practice, while Android does indeed transmit measurement requests as quickly as it can, most of the time many consecutive readings will return the same value. The explanation lies in the design of the fuel gauges themselves: for each metric, they contain a physical register that is updated with a certain frequency. If nothing (except the limitations of the serial link) prevents an individual from scanning as fast as he wants, the read data will be limited by this frequency, which varies with the model of integrated circuit. On the market, one can find refreshments around 4 and 10 Hz. It should also be noted that even if the `HIGH_SAMPLING_RATE_SENSORS` permission mentioned above were applied to fuel gauges, it would still be useless due to this inherent limitation of the hardware.

Finally, Figure 4.6 summarizes the evolution of Android's policy around sensors and on fuel gauges specifically, as versions go on.

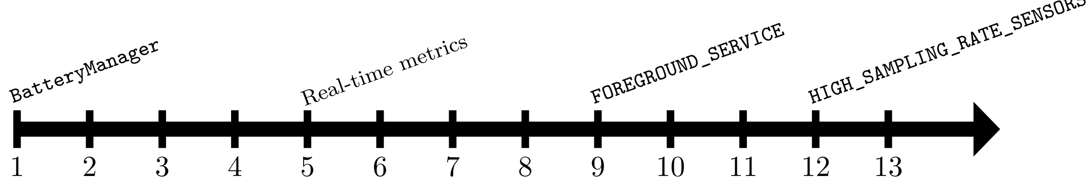


Figure 4.6: Historical timeline summarizing the events impacting the policy around fuel gauges, according to Android versions.

4.2.4 Identified risks

The simple fact that any third party application can access data from the fuel gauge without authentication and without requiring explicit permissions about it can already be a problem for the end user. If he disagrees with the such interactions, he cannot object to the sharing of power data. Moreover, when an application puts in place the technical means to retrieve this kind of information, even if it is for legitimate purposes such as energy saving, one can then question the real use that is made of it. For instance, the company Uber, which was the target of such suspicions in 2016, had to publicly deny this kind of exploitation [Cho].

These accesses being possible is the result of the lack of a moderation policy applied in any of the layers below the applications. In fact, it is interesting to recall that web browsers, while being applications, can also represent an additional layer themselves: they embed a Javascript engine capable of running such scripts and providing them a runtime environment. In it, an interface for querying data about the device's battery and power source is provided, called `BatteryManager`, just like Android's one. This additional layer, while including much less information regarding power than native ones, still does not contain any moderation enforcement regarding the frequency or the nature of requests. Thus, an updated version of Figure 4.5 could look like Figure 4.7.

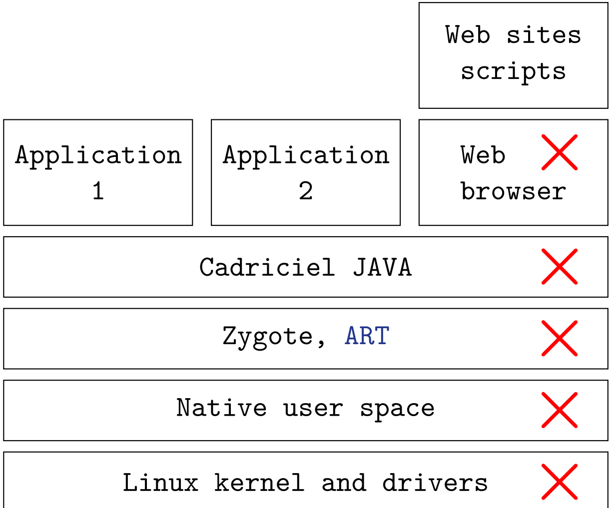


Figure 4.7: Updated simplified view of the abstraction model in Android systems. A red cross signifies that a layer do not implement a moderation policy preventing abuse with the help of fuel gauges.

As mentioned in subsection 4.2.3, fuel gauges have a hardware limitation preventing sampling at a high frequency. Disabling it from Android is impossible, and it is certainly justified for signal measurement reasons. In any case, this disqualifies, among others, attacks aiming at the execution of cryptographic code: the attack presented in [Koc96] requires, for example, measurements at the microsecond scale. The sample frequencies allowed by fuel gauges nevertheless leave malicious exploitations on human speed uses within reach.

Having regard to the state of the art exposed in section 2.2, and to the investigations done here concerning the security policy of Android about fuel gauges accesses, three threats are identified:

- The first threat is a privacy risk. An attacker can log, to the second, events such as the use or not of the phone, the activation or deactivation of wireless connectivity, the reception or transmission of communication, etc. Just like in many cases of spying or privacy issues, information that can seem to have low significance, like the time at which these events occur, or metadata, can actually lead to important and consequential deductions from an offensive actor.

- A second threat due to the creation of a covert channel on the platform: the real-time consumption tracking. Indeed, it is accessible in reading mode by all applications. But it must also be considered that all the actors have a *de facto* inalienable right to write on it, since each one can, by its execution, cause a lesser or additional consumption. Thus, for example, an application `A`, having access to sensitive data but not to the network, could transmit them, by means of certain signal modulation techniques, to an application `B`, having access to the network but not to the sensitive contents. The voluntary but unnecessary consumption can be generated by using any embedded component or accessory, or by making a large batch of useless operations, like matrix multiplications, for instance.

- A third threat, particularly aimed at implementations of secure and sensitive solutions: on many fuel gauges, the refresh rate, although low, is of the same order of magnitude as human interactions. One can then fear the harvesting of information during the entry of a secret data, or during any process ongoing at human-like speed.

In the framework of this thesis, a focus will be done on the third threat. Knowing the important state of the art in this domain, it becomes urgent to investigate the recovery of a PIN code intended for another process, on Android platforms. The devices based on this OS often are smartphones and tablets, and they do tend to host contents from many diversified contexts: personal and work lives, first party and third party developers. This is the recipe for attacks with painful consequences.

4.3 Sensitive data recovery through fuel gauges

4.3.1 Testing tools and experimental conditions

In order to demonstrate the existence of an attack vector, focusing on common, usual environments was paramount. It is indeed important to work on devices as they are used by any member of society. Thus, smartphones with classic Android images, or Read-Only Memories (ROMs), were considered, with versions ranging from 9 to 12. Targeting earlier versions of Android should not pose any obstacle, except for the lack of standard fuel gauge support prior to version 5. Both “stock” images and Original Equipment Manufacturer (OEM) ones were used. Stock ROMs are Android systems that have not been significantly altered and modified comparing to the Android Open Source Project (AOSP), which is the strict base of this OS. Some tweaking might have been done in order to ensure hardware compatibility, but the general features along with the UI should remain unaltered, and no third party applications should come packed with it. OEM images of Android, however, are the ones present in smartphones from third party constructors, at unboxing. They can be heavily different from the AOSP equivalent. A lot of content is often added at the application layer. The UI can be radically divergent, just as the Android framework in general. These modifications are not contained above the native layers: the Knox and DeX products, from Samsung and embedded in its machines, are both technologies not included in stock ROMs, that are living at the native user space and kernel space.

The vast majority of the works have been conducted on devices in the Nexus and Pixel lines from Google. These are popular, and often serve as models for other constructors, especially in the software domain. However, it should be noted that models older than the Nexus 6 and Nexus 9, or more generally, that were released prior to 2014, very rarely embed fuel gauges in them.

Tests have been executed both with and without the USB power source plugged. When the fuel gauge is not connected to a power source, it returns negative values for the instantaneous current consumption, as energy is being drained from the battery. However, when a charging cable is connected, positive values are obtained if the system consumes less energy than it absorbs

via the cable. In the conducted experiments, it was observed that the charging current is stable enough not to question the attack.

To conduct the temporal attack, a simple target application was developed. A screenshot of its interface is shown in Figure 4.8. It prompts the user to enter a PIN code, using Android's native virtual numeric keypad. As always, the entry must be confirmed using a validation key located in a known location on the screen, here in the bottom right. Additionally, one can notice that the keypad is not perfectly square, with varying distances between certain keys, which can also slightly work in favor of an attacker. It is worth noting that most phones come with a factory default configuration that includes a vibration feedback every time a key is pressed, which requires substantial energy to activate, representing an aggravating factor in this situation. This is notably the case on all tested stock Android ROMs, and on all tested smartphones and tablets from Google's Nexus and Pixel lines.

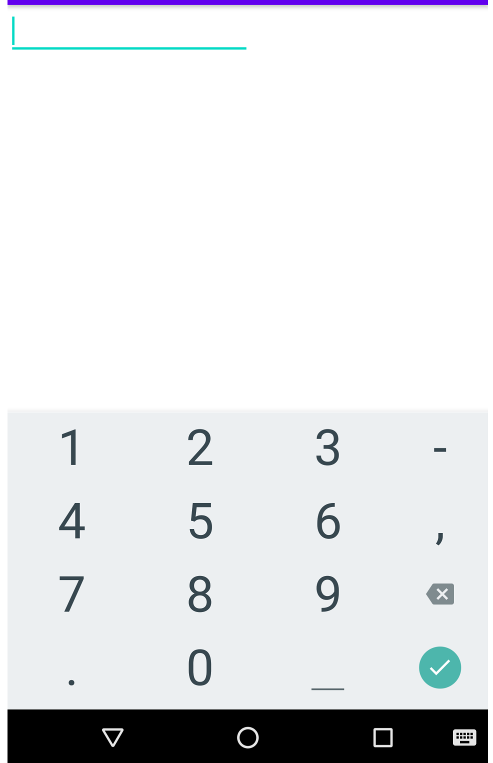


Figure 4.8: Screenshot of the target application.

An Android application dedicated to the attack was also developed. The application includes a service that can scan the fuel gauge without interfering with the user interface and can do so even when the phone is in standby mode or when the user switches to another application. As energy readings accumulate, they are stored in memory and can be accessed later, to avoid lowering the signal-to-noise ratio that could occur with real-time extraction via a wired or wireless Android Debug Bridge (ADB) link. For the purposes of these experiments, a graph will be directly displayed on the smartphone, showing the data collected from the fuel gauge. The x-axis will represent the time in milliseconds, and the y-axis will indicate the selected measurements from the fuel gauge.

4.3.2 Data recovery

To spy on a PIN code being entered using the fuel gauge, power data must be polled during the process. Several types of information are available and provided by the `BatteryManager` Android system service. The capacity, despite its name, actually refers to the state of charge. It is given as a percentage, without fractional part, it is thus clearly not precise enough. The average current gives a value in microamperes, representing the current going in or out the battery in average, over a period of time determined by the hardware, i.e. by the fuel gauge itself, that can not be changed. The accuracy could potentially be satisfying, but the average over a considerable period of time makes it unsustainable for this attack. The charge counter and energy counter are two good candidates. They respectively return the battery capacity in microampere-hours, and the battery's remaining energy in nanowatt-hours. The units used suggest a good precision. If their hardware refreshing rates are good enough, and if they are estimated properly, they can be a good fit. It is highly probable, however, that one of the two is simply mathematically derived from the other, meaning it would be useless to use the two at the same time. However, the last possible measure that the service can return, the instantaneous current, is the best available, and is the one that will be used for the attack. It is delivered in microamperes, and, on the vast set of considered fuel gauge models, always has the best hardware refreshing rate.

The Android application developed to operate the data recovery will thus print the instantaneous current consumption on the y-axis of its graph. In order to make sure to polling of the fuel gauge does not get killed by Android when the user switches between different applications, it must be implemented in a service, like any compute-intensive task that could impact the UI otherwise. Requesting the `BATTERY_PROPERTY_CURRENT_NOW` value from Android is done by soliciting the `BatteryManager` system service just like any other, as shown in Listing 4.3.

Listing 4.3: Java code for getting an instance of the `BatteryManager` system service, and for querying to it the instantaneous current consumed by the platform, in real time, on Android.

```java
BatteryManager mBatteryManager =
   (BatteryManager) this.getSystemService(Context.BATTERY_SERVICE);
long courant =
   mBatteryManager.getLongProperty(BatteryManager.BATTERY_PROPERTY_CURRENT_NOW);
```

As a first step, it is required to see if user interactions can be detected. To determine this, the polling of the attacking application must be enabled, then the screen must be touched, preferably on other activities. Figure 4.9 contains the result of an example sequence, where a finger is resting on the touchscreen panel between 12,500 and 22,500 milliseconds, counted on the x-axis. The increased downward variance in this region indicates that the fuel gauge are well able to detect the delta in power consumption of a phone or tablet when touching the screen, even without vibration. Using a device with a cracked, but functional touchscreen does not change this result. The curve decreases when touching because the consumption increases during these moments: there is more current coming out of the battery. While it is likely that this consumption differential is due to the physical phenomenon at play on capacitive technologies, one can also assume that a software processing necessary to manage this input mode is also responsible. This result allows continuing research in this path.

The next tests are to simulate an actual offensive attempt at a PIN code entry. The fuel gauge polling is enabled on the attacking application. Then, on the target application illustrated in Figure 4.8, the numbers are typed in, for example here, 1, 4, 7, and 3, along with the confirmation key. When returning to the attacking application, a result that can be printed on screen is possibly the one illustrated in Figure 4.10a. In the middle of the graph, one can visually notice five downward peaks. These are the five consumption peaks corresponding to the five keypress: the four numbers, and the validation key. Figure 4.10b shows a zoomed-in view

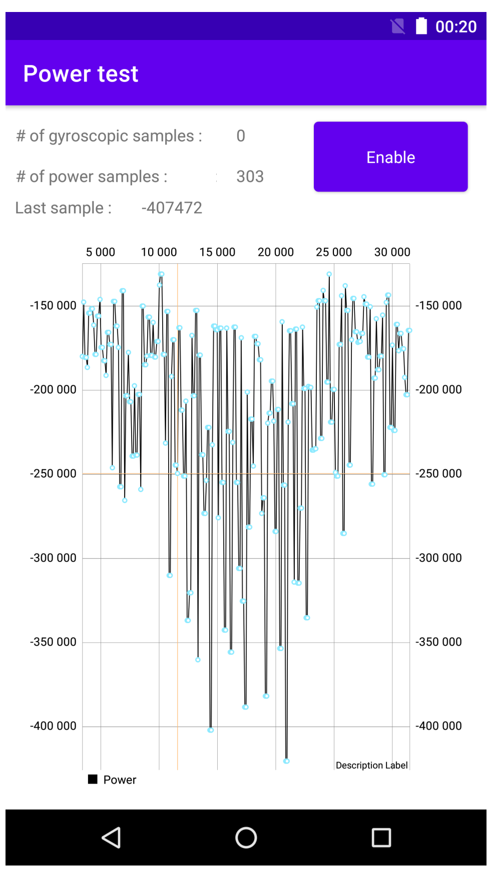


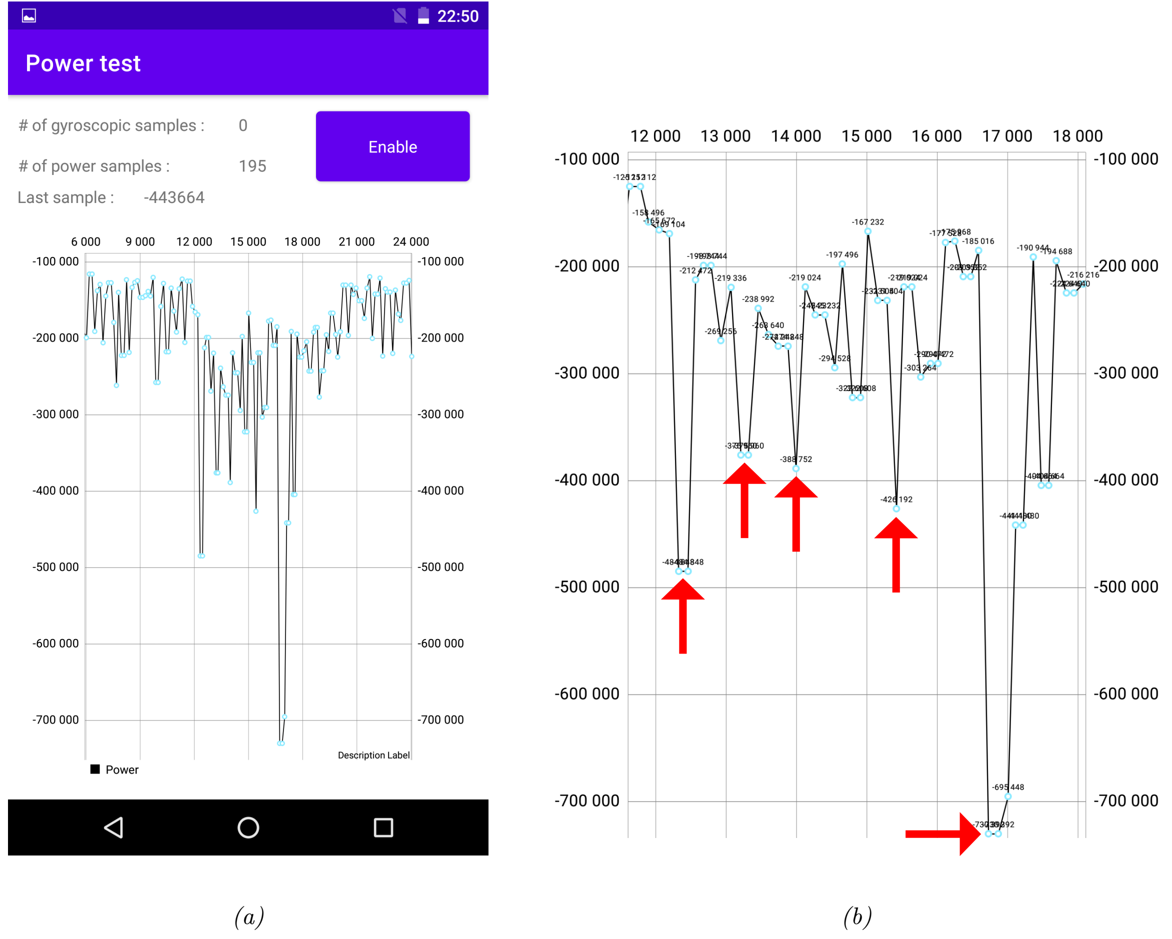
Figure 4.9: Screenshot of the results obtained by the attacking application, when trying to see if screen taps are detected. The x-axis represents the time in milliseconds, while the y-axis shows the consumption in microamperes. Here, the screen was touched between 12,500 and 22,500 milliseconds.

of the graph, with the peaks being highlighted.

The collection of such results tend to indicate that, as an attacker, one could go on the next step, meaning that he could proceed to data analysis. However, if the collection happens on a fuel gauge with a hardware refreshing rate that is too low, some additional accuracy might be needed. In this case, enabling the collection of gyroscopic data should help. Figure 4.11 shows the combination of gyroscopic data and power data. One can notice that gyroscopes can be polled at a much higher frequency than fuel gauges. In the face of such data, an attacker should focus on the temporal points where the three gyroscopic axes have a null derivative, just before the consumption peaks. Indeed, the stabilization of rotational movements are indicative of the impact of a finger against the screen. They must be particularly considered before consumption peaks because fuel gauges, even when asking for the instantaneous current, return the current consumption measured on a small time period. Each measurement thus represents the current on the last small window, just before.

4.3.3 Data exploitation

The native virtual keypad provided by the Android OS always has its validation key at the same location, in the bottom right of the screen. Thus, when the user types a four-digit PIN code, an attacker will necessarily know where the fifth touch is located. The whole goal of the power data exploitation is to infer information about the secret code, by using the delays between the taps, and the fact that the position of the last one is known. It should be noted that the absolute time points where the taps are do not matter. What is actually helpful in this attack are the



(a) (b)

Figure 4.10: Screenshots of an example of what can return a fuel gauge during the entering of key presses. The x-axis represents the time in milliseconds, while the y-axis shows the consumption in microamperes. Figure 4.10a is the whole screen, Figure 4.10b is a zoomed-in view of the same data, with the five peaks being highlighted.

delays between them; more importantly, the ratios between these delays. This can be clearly observed on Figure 4.10b, which presents the trace obtained when typing 1, 4, 7, and 3. The delays from 1 to 4, and then from 4 to 7 seem identical, which is coherent, since the distances from 1 to 4 on one side, and from 4 to 7 on another, are the same on the screen. The delays from 7 to 3, and from 3 to the validation key, however, are longer, which is coherent too, since these distances on the screen are, as well.

Once the four delays between the five taps have been determined, there are many, different ways of inferring PIN secret codes based on them. In [FKK10], the possible sequences are extracted with the help of a Hidden Markov Model (HMM). In [PLHQ20], the use of machine learning techniques was highlighted, however these require training before being applicable on victims.

Here, a deterministic algorithm based on recursion has been explored. Depending from the time lapses provided, it unrolls the tree of the possible codes. It starts from the end, meaning the validation key, since it is the anchor point used to get location information. It then develop up to the first number. It works in reverse order, thus the first numbers of the code will be more uncertain than the last ones. The way it works it by inferring the most likely previous key touched, based on the corresponding delay with the current key. If it is small compared to other delays, then it is likely a near key. If it is large compared to other delays, then it is likely a distant key. When multiple keys are at the same distance, the tree will spawn multiple

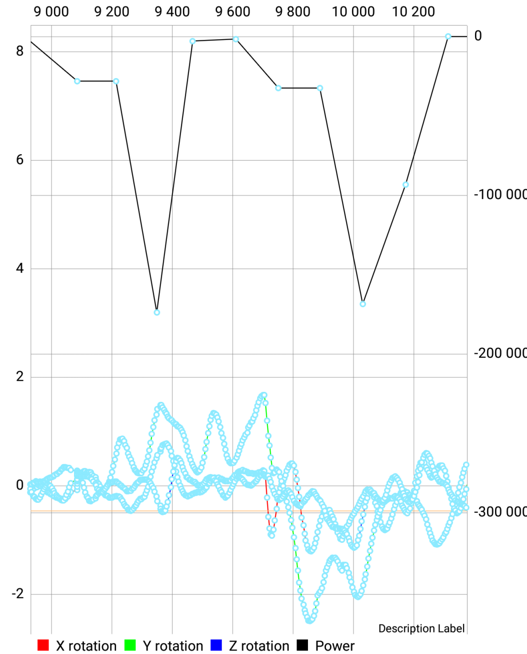


Figure 4.11: Capture of a polling result graph where the gyroscopic data, in blue points on the bottom, has been superposed to the power data, in black at the top. The x-axis represents the time in milliseconds. The left y-axis is dedicated to rotation rates in radians per second, while the right y-axis shows the consumption in microamperes.

branches at this level, since it is impossible to tell which of them the user has touched. This is why multiple PIN sequences are proposed at the end of the algorithm. Figure 4.12 shows a graphical representation of the tree one might get with such a method. All of its branches end at the validation key, but can have radically different starts. The method presented here is conceptually close to the one presented in [CCBG19].

It should be noted that a very powerful aspect is that the Android native virtual keypad is rectangular, and does not have a square shape. Indeed, its height is smaller than his width. The horizontal distances between its keys are longer than the vertical distances. For instance, the distance from 1 to 4 is considerably smaller than the distance from 1 to 2. This fact is useful to attackers, as it significantly helps reducing the most likely paths as the algorithm is running, and thus the number of resulting PIN codes returned at the end. It might, however, require spying more entry sequences, to refine the measured data.

The presented algorithm has been implemented using Scheme, a dialect of the Lisp programming language. This choice has been motivated by the fact that developing the tree of the most likely PIN sequences is a heavily recursive operation, and languages in the Lisp family provides an excellent efficiency in these, as they rely on functional and procedural programming. Listing 4.4 shows the output obtained based on data similar to the one exposed in subsection 4.3.2, but with perfect accuracy, which can be approached with a high enough number of captures. It contains two proposed sequences. These are in reverse order, and the 10 key they are starting from is actually the validation key. Thus, here, the first proposition is the 7473 code, for instance. The second one, which is 1473, is the correct one. They are not ordered following any correctness probability; in fact, considering only the data that is available, one can only admit all PIN code propositions are equiprobable, since the corresponding distances between their numbers are all the sames. Also, it should be noted that for this kind of program to work, it must be fed with the geometrical data of the Android native virtual keypad, meaning the distance from every key of it, to every key of it.

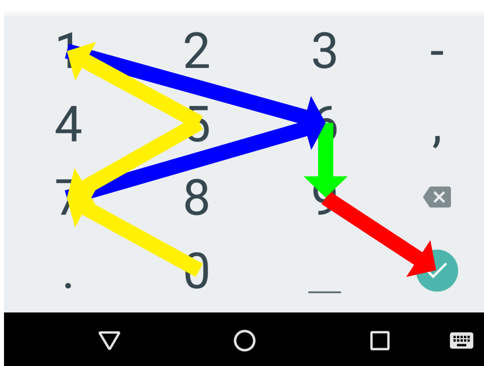


Figure 4.12: Graphical representation of the tree that can be developed, on top of the native Android virtual keypad, invoked by the target application. The red arrow represents the first most likely path, starting from the validation key. The green arrow represents the second most likely path. The blue arrows represent the third most likely paths. There are two because the 6 key is at the same distance to 1 than to 7. In the same way, the yellow arrows represent the fourth most likely paths.

4.3.4 Discussion about the results

Overall, this attack demonstrates the vulnerability of Android devices to temporal attacks, which can be exploited by an attacker. The required knowledge and resources to do so are accessible to private individuals, and being an organization with large means and funding is not necessary.

Estimating the success probability of these offensives and the average number of PIN sequences resulting from it is not simple, because these values deeply change depending on the valid keys. For each possible code, they are different, since the geometric behavior between keys poses different uncertainties. In general, PIN sequences with good variability, meaning non-uniform distribution across the screen, provide good results for the attacker, thus bad one for the end users. When good traces have been acquired, it is very frequent to get less than 10 propositions, or even less than 5 sometimes, for PIN codes containing 4 numbers. In fact,

Listing 4.4: Example output from the developed deterministic algorithm. The 10 key represents the validation one, pressed at the end of the sequence. The proposed sequences are in reverse. The first line is the command, the second line is the output.

```
(arbre '(3.37 4.01 1 1) (cons '(10) '()) 0)
=> ((((((10 3 7 4 7) (10 3 7 4 1))))))
```

exploiting the data will not be the main challenge; it will rather be to get good real-time consumption curves, and as many as possible. Satisfying traces are characterized by signals with visible peaks and no noise from parallel ongoing operations such as automatic screen brightness adjustment or sudden wireless communications for example. They should represent screen touches done in regular conditions, with the user entering his code in a common, carefree way. Obviously, if he is aware of any temporal spying and tries to interfere with it, the whole process is not viable anymore.

As a reminder, the temporal analysis does not work with the absolute times between current peaks, but with relative ones in comparison to the other distances in the same PIN sequence. A user can thus type his code faster or slower without disturbing the attack's conduct, as long as it is in a regular manner. Hence the possibility to exploit multiple consumption traces when looking for only one code.

Contrary to what one might think, targeting codes containing more numbers does not automatically lead to getting more proposed sequences. The relation is not even proportional, and, in fact, the result count is often similar when dealing with 4-number codes than with 6-number codes, for instance. This is because developing a tree of possible sequences with a high number of keys could potentially generate more possibilities, but it also has to take into account more constraints, as there would be more distances to abide to. All in all, it leads to long PIN sequences actually not being impossible to spy on.

During data collection, it was shown how gyroscopic data can be used to get better temporal points. If an offensive actor is not satisfied with the number of PIN sequences obtained, he can use the gyroscope in another way too. Instead of refining the timings, one could conduct a kinetic study in order to discriminate between the considered codes. Indeed, for the same end user, tapping the 1 key will require different rotations than to type the 9 key. This is because in order to have his thumb touching the right location, the phone will have to be tilted in different ways. This fact can be particularly useful for the attacker: if the gyroscopic data have been captured with the power traces, the kinetic study can greatly help in determining the most likely first key of the sequence for instance, thus achieving a probability rank in the obtained PIN sequences. If the targeted secure system enforces an entry limit before locking (often around 3 or 5 tries), this can proves to be truly helpful to offensive actors.

In general, and outside the scope of PIN codes, this work puts into question the confidentiality of platforms based on Android. Indeed, while fuel gauges provide a comfortable delegation of responsibility, they also compromise the security of the whole device if implemented improperly. Considering the possibilities shown here, any developer should be careful when producing content targeting Android, as many threats could be conducted against it, beyond PIN entry attacks: general spying, information extraction in secret, and others. All these threats do not require modification of the Android image, nor do they require rooting the device in order to get administrative rights. Thus, this is not a hazard linked to a minor fraction of the global fleet of products. This concerns all devices, and pleading that this is a negligible phenomenon is not possible. A noticeable exception, however, might be reinforced Android ROMs like the one based on GrapheneOS [Gra], that have more strict security policies. Nevertheless, they are installed on an extremely small fraction of devices.

This security concern does not only impact third party developers, it also affects Android itself. Indeed, the spying of PIN entries explained above also work on the unlocking screen. This is because the polling is implemented in a service, which, most of the time and depending on the Android images, would not get killed for reasonable periods. Being able to get information on the user's PIN code is a heavy breach, with large and numerous consequences. Today, it might be considered that PIN codes are an obsolete first authentication factor, but:

- they are still used that way by some people, especially the ones with old smartphones and tablets not equipped with biometric sensors.

- they are still used that way by many private services, notably in banking (where this is the most popular CVM) or others (the Carte Vitale application relies on PIN codes in order to be unlocked, and the France Identité application requires to type on a virtual keypad numbers from one's identity card).
- they are still a major second factor when the first one does not work. If the latter is based on biometrics, this is common in case of masked faces or dirty fingers.

Hence, the risks concerning PIN passwords should not be ignored. However, the exploitation of fuel gauges, as they are today, does not allow the collection of good enough information when typing on a full keyboard containing letters, like the ones based on the QWERTY and AZERTY layouts, because of the refreshment rate being too low. Spying on pattern entries does not work, because the user will keep its finger on the screen during the whole drawing process, and will only take it off at the end. The resulting graph would be a long sequence of overconsumption, without any interpretation possible.

When targeting the Android unlock screen, an interesting fact for attackers is that some applications, under some conditions, can trigger the locking of the device. These conditions depend on many factors including the Android version, image and root status, and often contain the requirement of having some administrative rights. But if available, this option creates an attractive offensive scenario, where an application providing a video game would actually be a Trojan horse scanning the fuel gauge in the background. The video game could voluntarily be locked in an inopportune moment, thus encouraging the user to type his PIN code to quickly get back on the game. This would result in a convenient power trace probably lasting around 5 seconds.

Finally, in the framework of this thesis, a focus has been done on Android platforms, as they are quite common and now embed a lot of personal data. But the bad implementation of fuel gauges and improper security policy about them can theoretically impact any system having one. Video game consoles, for instance, can integrate such circuits. One unstudied example is the Nintendo Switch, which is based on FreeBSD, and embeds a MAX17050 from Maxim Integrated, as shown in Figure 4.3.

4.3.5 Countermeasures

What can be done against information leaks from the fuel gauge? Countermeasures are available on different abstraction layers, and depend on who is asking: third party developers, or engineers at Google.

The attack presented here works when the end user is typing his code with one finger. He could mitigate the offensive process by typing it with two fingers, for instance. Putting the smartphone on a flat surface before entering the code would be beneficial, too, as the gyroscopic data, that can be used to help, would not be useful anymore to the attacker. However these measures place more responsibility on the users and are quite inconvenient. The everyday use of an Android device with many sensitive contents in it should not require such precautions.

As mentioned earlier, some PIN codes are a little easier to find than others, because sequences with a non-uniform distribution of keys across the screen help the algorithm developing the tree of possible combinations, as it will provide less possibilities at each step. However, this phenomenon has not been seriously studied, and even less theoretically modeled. It is thus hard to give any advice to end users about the selection of a good PIN code in regards to this consideration. And, in any case, this type of countermeasure, just like the previous ones, wrongly puts the burden on the shoulders of end users, who should not have to endure this when using their personal smartphone. Also, the space of possible 4-number passwords is already very much small; adding constraints on it, and thus further reducing it, cannot be advisable.

A simple measure that can be implemented by third party developers is the use of a randomized keypad. At each invocation, it would appear with each key being at a new, random location.

In this case, an attacker would not be able to do any distance estimation between taps, because he would not know where each number is. The development of such a solution is not heavy nor onerous, while being radically effective against this offensive attack. However, it harms accessibility, especially for visually impaired people, who will have a hard time determining where each number is. Also, as such a keypad would not be the native one, it would goes against the principle of reusing available code and utilities, which is favored in software development, as it creates good specialization and optimization, and to avoid the recreation of misdesign and security errors previously corrected in the past, which is common when reinventing the wheel like this. This solution, however, is quite common in the banking domain.

Something that can be implemented by third party developers not able to modify the system is signal jamming. When soliciting the user for a sensitive operation, like a PIN code entry, an application could voluntarily trigger useless tasks, in order to make the overconsuption of tapping the screen not readable anymore. The technical implementation and details of such a solution can rely on many different things. Sensitive signal simulation could also be considered, where a signal looking like the one produced during a PIN entry could be crafted by the application. Later, if it gets a passcode corresponding to the fake signal, then it would know it is under attack by someone spying on it, and could trigger any defensive measure in consequence.

At the system level, developing a trusted UI system, like the one presented in [FPP23], can be a long, costly process. And it will likely not prevent such leak from being damageable. Indeed, if applications in the REE are still able to freely access power data, then it does not matter if the user types his PIN code on a highly secure interface: he will still tap the touchscreen, which will still trigger overconsumption on the instantaneous current graph, probably because of both hardware and software reasons.

If one has access to the system level, and is able to modify it (whether it is by being an Android developer able to modify the source code, or by being able to modify the system at run time), then patching it becomes evident. Correcting the `BatteryManager` system service is easy, and would only require a few lines of code, essentially modifying Android's security policy about fuel gauges. But in reality, third party developers never get these kind of accesses. And modifying the default permissions of the Android OS would inevitably result in a lot of legitimate applications not working anymore, as they took these accesses to power data as granted.

Synthesis

In this chapter, the context around the management of portable energy sources is recalled, along with all the challenges it brings in safety and comfort of use. Fuel gauges, which are ICs dedicated to the monitoring of batteries, are introduced.

It is described how they are more and more implemented in smartphones and tablets by constructors, in order to ease these aspects during design. The security policy Android holds against them is detailed, and in front of how tolerant it can be, several different risks are declared: user spying, secret communication channel establishment, and spying between applications.

This chapter provides a focus on the risk of an user having his PIN code spied on while he is typing it on a legitimate application. It is shown how, by using fuel gauges, one can get considerable amounts of information about it, to the point of getting few possible sequences. This security hazard is discussed, along with several countermeasures.

Part III

Conclusion

Chapter

Outcomes

Abstract

This chapter concludes this manuscript. First, a recontextualization is done, recalling the omnipresence of personal COTS devices in today's societies, their power, and how they have more and more responsibilities. The different ways of deploying secure solutions on them are examined, along with their consequences. Then, the contributions presented in this manuscript are reminded, both concerning the McEliece cryptosystem, and the security of power management in Android. Their impacts and what they help to realize are explained. Then, all the possible future works that could consecutively take place are listed, whether they already started or not. Finally, a two-part analysis is developed: first, on the subject of the industry and competition. Secondly, on societal implications.

Chapter content

5.1 Analysis **79**
5.1.1 Recontextualization 79
5.1.2 Synthesis of the contributions 81
5.1.3 Impact of contributions 82
5.2 Prospects **82**
5.2.1 Future works 82
5.2.2 Industrial and competition considerations 83
5.2.3 Societal considerations 84

5.1 Analysis

5.1.1 Recontextualization

In the last decade, the population of developed countries became more and more equipped with smartphones. In France, for instance, the ownership rate amongst young people was near 95% in 2021 [SAL22]. Also, these machines have been integrating more and more powerful hardware over the years. To give an idea of this increase, the specifications of Samsung's flagship smartphones in the Galaxy S line can be looked at. In 2014, the Galaxy S5 embedded an Arm big.LITTLE architecture with 4 Cortex-A15 cores at 2.1 GHz and 4 Cortex-A7 cores at 1.5 GHz, along with 2 GB of Random Access Memory (RAM). Ten years later, the Galaxy S24 aimed at the European market, which is the least powerful version, embeds 1 Cortex-X4 core at 3.2 GHz, 2 Cortex-A720 cores at 2.9 GHz, 3 Cortex-A720 cores at 2.6 GHz, and 4 Cortex-A520 cores at 2.0 GHz, for a total of 10 cores, along with at least 8 GB of RAM.

The direct result of these phenomenons is that on the side of states and companies, it became more and more acceptable to simply consider that the whole population is in possession of a powerful and available smartphone. Hence the incentive the switch to the digital alternative of many services: payments (by tokenizing one's payment card), identification (by importing

identity or health insurance cards), transport (by using digital tickets), theft protection (by using digital versions of one's house and car keys), and so on.

Beyond the feeling of a better efficiency in everyday life, this represents a major paradigm shift. Up until now, the electronics in charge of providing such services used to be under the responsibility of the service providers. A payment card is produced by a person's bank and issued to him: the manufacturer took the necessary measures to make sure nothing could disrupt the operating of the Java Card applications inside dedicated to payments. Besides protecting his PIN code and keeping the card personally, no requirement ever was demanded to the client. The same can be said about identification, transport, and others. This is true both in terms of security (nothing is requested about the client's personal effects) and in terms of financing (the hardware is provided, the user's choice about one device or the other does not occur, and cannot have any effect on safety). However, when service assets are digitalized, the users' free will deeply impact how things operate. Did a person root his smartphone? Then the ability to pay with it is likely lost, along with any possibility to receive payments. Here too, other services are impacted as well: the France Identité, Bonjour RATP, and Netflix applications will not fully work on devices with administrative rights unlocked. Installing an Android ROM different from the OEM one will often bring similar troubles. Beyond the considerations about the end users' behavior with their devices, inequalities depending on products and the people's ability to afford for high-end ones exist. Cheap smartphones frequently lack hardware security mechanisms that expensive ones can provide, a emblematic example being SEs. This inevitably leads to a society where members with relatively good incomes have more features, or features better secured than the others. In comparison, electronics provided by banking or transport operators, or the state itself, did not imposed any behavior or financial investment, and this equally to all citizen.

To tackle all system inequalities and versatility, many service providers have investigated around the white-box security model, where everything his happening under public scrutiny. This can possibly imply many different approaches. One that is particularly widespread is the use of turnkey protection solutions, most of the time commercialized by third party companies. They usually provide measures like obfuscation and instrumentation blockers, claiming to make products unexploitable to attackers. Nonetheless, they are based on techniques that are necessarily oversteppable, and thus mostly provide guarantees about the required time for an offensive actor to get what he wants. Another approach, which is white-box cryptography, as introduced by Chow *et al.* in 2002 [CEJP03b, CEJP03a]. But while it might be based on sophisticated mathematical techniques, and might be able to hold quite some time, it still suffers from the same issue as turnkey third party software. Yet another approach to the white-box security model is to design from the ground up a solution so that running under open scrutiny is not an issue to start with. This is regularly the chosen path when HCE is exploited: since the use of a SE has been discarded, sensitive assets will likely be stored online, in a private, secure infrastructure. The next logical step is to, for instance, distribute perishable secret tokens to the smartphone, from the online secure storage, with limited associated value, so that by the time an attacker has got his hands on them, they are either obsolete, or only allow limited damages. In practice, when a service provider makes an attempt toward this approach, it does not dare to store sensitive assets like tokens in clear form and in the REE, so the use of obfuscation and other similar techniques like mentioned before are involved, here too.

If a service provider would happen to implement a perfect white-box solution, no hardware security mechanism would be required, and compatibility across the whole fleet of Android devices could easily be guaranteed. Hence the massive efforts towards such methods, despite the fact that, in reality, this paradigm is considered only good enough to bring temporary security. This concern is often addressed by the industry with frequent rotations of sensitive assets like cryptography implementations or tokens, leading to a level of risk they deem acceptable.

5.1.2 Synthesis of the contributions

Because the white-box model relies heavily on the amount of work required to attack an implementation, the notions of time and efforts become paramount in how security is conceived and achieved. If any offensive technique allows to circumvent the barriers making exploitations longer and harder, then the very premises of this paradigm fail. Among theses barriers, mathematical methods, obfuscation, and isolation features from the environment can be considered. However, in this thesis, it was shown how fault injections can compromise implementations of the McEliece cryptosystem potentially based on obfuscation, and how inter-process isolation in Android can be breached because of available hardware resources.

In the first case, in Chapter 3, the original McEliece cryptosystem definition and specification were given again, and the advantages it can still offer today were detailed: notably how it is recognized as robust in the scenario of post-quantum attacks, how it is efficient, and how it can easily be further accelerated. It was recalled how it was the target of numerous cryptanalyses when error-correcting codes different than the Goppa ones were used, and of offensives based on side-channels. However attacks based on fault injection were not targeting the secret key. The rest of this chapter then proceeded to actually disclose one such attack discovered as part of this work. Instead of targeting the specification of the cryptosystem, it was decided to target a common way of implementing it, specifically a way of using the permutation matrix. By replacing one specific instruction by another, an attacker can get information about this matrix, which compromise the security of the secret key, and can be a threat to obfuscated implementations: this instruction switching, available on the Arm architecture, is described, in support of this specification's instruction format and operation codes. The details on why this offensive was designed that way, its constraints, its capacities, and how to proceed with it were given, along with the results one can obtain. Since the methodology for this attack is quite versatile depending on many different parameters, two step-by-step completely illustrated examples were furnished, with detailed explanations to make it more intelligible. In the end, since there is a gap in the propositions of post-quantum algorithms usable in an open, uncontrolled context, propositions of modifications were suggested to make it more compliant to the white-box scenario, based on actual precalculation techniques, rather than turnkey simple obfuscation. These impact not only the private key side, but the public key one too, necessarily.

In the second case, in Chapter 4, the omnipresence of Android in COTS products dedicated to hold a lot of sensitive data about people's personal life was recalled and explained. Since this OS is mostly deployed on portable devices, a core concern about them was analyzed: the management of power and its stakes, which are mostly about safety, life time, time of charge, and good estimations. Since lithium batteries are the go-to of portable energy sources these days in smartphones and tablets, a detailed technical explanation about how they are hard to master and quite unpredictable was developed. It was then considered how, recently, more and more constructors delegated all these considerations to fuel gauges, which are small ICs they integrate on their platforms, in order to have it monitor the battery and make the complex predictions about it, for them. How they are physically implemented was studied. Then, in order to understand how Android integrates fuel gauges in software, an assessment of how this OS is structured and how it handles interactions between applications, and interactions between applications and hardware resources was made. A focus on what Android allows or forbids applications to do specifically regarding fuel gauges was developed, which led to the declaration of several identified risks for these platforms. One, in particular, was greatly detailed: the risk, for the end user, of having his PIN code spied on while he is typing it on a legitimate application. Indeed, fuel gauges are precise enough, and Android so lenient about them, that such a procedure is conceivable. Thus, the methodology an offensive actor can follow was described, both to recover the necessary data with the help of fuel gauges, and to analyze it properly thereafter. The rest of this chapter was dedicated to several discussions about this risk,

and to the listing of possible countermeasures in different abstraction layers, along with their corresponding advantages and disadvantages.

5.1.3 Impact of contributions

These results expose the reality and the *status quo* of trying to run sensitive content on open REEs. Theoretical and mathematical methods, obfuscation, along with integrity and anti-instrumentation measures allow to get ahead of offensive forces for some time. When they have come up with ways of circumventing them, new measures can be applied, and so on. These results also reflect the complexity of extending a RoT. These trusting anchor points are often materialized with SEs, TEEs, TPMs or any other hardware security feature in COTS devices. In order to extend the trust they benefit from to the REE, integrity verification and attestation are frequently implemented: the software and data state of the open environment is, to some extend, verified by the RoT. This approach can indeed be efficient: as a matter of fact, it can help in countering attempts at reproducing the attack on the McEliece cryptosystem presented in this thesis. This is because if all the software tools required to simulate a fault injection are absent or disabled, and if this is verified and confirmed by the RoT, then proceeding to the attack without hardware injection will inevitably be a lot harder. On the other end, the second main contribution of this thesis, which relies on spying through power metrics, shows that trust extension attempts over the REE, even if done perfectly, can also be irrelevant: a legitimate platform with verified integrity will still allow spying over PIN code entries. If the environment that is considered valid is guilty of flaws, then these flaws will be approved too. The works presented in this thesis are the illustration that computer security is a chain with many sensitive links: security of specifications, security of implementations, security of the compilation and runtime environments, reality of the targeted security model... All of them can potentially be the weakest link.

5.2 Prospects

5.2.1 Future works

The results obtained in the framework on these efforts call for multiple, different enhancements and further works. They can be subdivided into three categories, depending on if they are related to the McEliece cryptosystem, the management of power on Android, or the architecture of security on complex devices.

Regarding the McEliece cryptosystem, one judicious future work is to port the conducted investigations from the original specification of 1978 to Classic McEliece, the candidate of the NIST contest for quantum-secure algorithms. Here too, it might require to look after implementations rather than on the specification. A crucial point is that Classic McEliece does not use permutation like in the original specification: the cryptographic confusion is brought differently, meaning a simple transposition of the attack presented here will not be possible. Other paths of research include the improving of the proposed white-box measures. The decoding operation during decryption was here considered as unprecalculable. With further studies, precomputing it might be achievable. If this goal is reached, white-box implementations of the original McEliece cryptosystem would be much more efficient, and maybe more secure. Finally, improving the presented attack as is could be done: only one bit change in the operation code was considered here in order to keep the hardware fault injection scenario viable, but in a purely software context, bigger and more specific faults could be considered, as they would be easier to deploy. Without any doubt, this would result in much more powerful attacks.

On the subject of the secure management of power in Android, all proposed countermeasures mostly involve industrial and commercial choices, rather than technical arbitration. Fixing

the security policy is easy for system developers, but impossible for application ones, and will especially break existing, non-updated executables. Measures like keypad randomization are relatively easy to deploy and develop at the application level, independently from the system, but have downsides regarding accessibility and code reuse. Obfuscation of power consumption and fake sensitive signal generation are hard to implement, but can be independent from the system too. Choosing one technique over the others is mostly about which sacrifices the service provider is ready to make. However, while patching Android's `BatteryManager` and creating keypads randomly would not need any research works, the voluntary inferences on power signals could be studied more. In the end, the most promising works in this contribution field would be to keep investigating on the power management of portable devices. An interesting fact is that fuel gauges can now embed small cryptographic calculators, in order to authenticate the battery at boot time. This feature is mostly used to make sure the battery comes from a legitimate, validated source, but it could be studied in order to determine the possibilities around it, and its consequences.

The research axis on power in Android unveiled a larger issue than just securing the API to get information about the battery. It suggests a third, broader future work: the securing of complex systems, not only running executables from different third party sources, but also embedding components from different third party sources and making them collaborate. The breach in Android presented here can be considered as the result of integrating a class of components without realizing it would impact security on another place of PCB, namely the SoC, so to speak. Finding the other inter-component, unnoticed and unwanted interactions is paramount, as they can take unexpected forms. A Systemization of Knowledge (SoK) paper is currently under writing. It would help in raising awareness against different surprising relations, and in the modeling of secure components assembly. Being able to anticipate any such risk upstream, during the very conception of devices, would be ideal.

5.2.2 Industrial and competition considerations

Software living in trusted environments benefit from many strong, lasting guarantees. In comparison, how white-box solutions are not ideal has been extensively covered in this thesis. The main consequence of this it that the gap between developers having accesses to SEs or TEEs, and other developers who do not, can be considered substantial, instead of insignificant. It can be seen as the essential contemporary division between providers, especially on smartphones and tablets. On one side are multinational companies which are constructors of components or devices, and on the other are approximately all the rest of actors, companies or not. Having verified in practice some of the actual issues white-box implementations can encounter in COTS products, ignoring the various impacts they have would be unfortunate.

This state of things raises many questions about fair competition, and control on people's devices. Implementing secure systems is highly complicated for third party service providers, whereas the companies with additional hardware accesses can benefit from strong protection models. In consequence, the ability of industrial actors to compete in this field will, in most part, not be determined by their efficiency and expertise, leading to unfair competition. Getting to a situation where actors could get similar chances in securing their systems is technically possible. Installing new, third party content on SEs is possible, as long as the content is signed and benefits from a certificate. But occurrences of this actually happening in products are extremely rare. However, a good example of industrial dialogue to share secure hardware resources is the certification of software in the framework of secure boot. On PCs, to avoid rootkits, every early-stage software loaded during booting can have their integrity verified, and validated against certificates embedding public keys. These are necessarily deployed during manufacture of motherboards. There cannot be a large number of them: in order to keep collaboration with motherboard constructors easy, there should only be a few. In practice, Microsoft created one

that can be used by third party OS developers, in order to benefit from secure boot. It is called Microsoft Unified Extensible Firmware Interface (UEFI) Certification Authority (CA). Any actor can potentially ask for Microsoft's certification; once obtained, their boot stage can successfully be verified and run with this feature enabled. Microsoft's participation is paramount since the integration of certificates in PCs' hardware is up to motherboards constructors' goodwill, thus, having some industrial weight is important. The certificate of Canonical Ltd., in comparison, has much less coverage, for instance [Jer13]. In the end, these technical choices allow potentially any actor to benefit from secure booting, which is an efficient and reliable security measure relying in some part on hardware. The end user also has the possibility to get any arbitrary content signed using personal keys, or to remove institutional keys, like the ones of Microsoft or Canonical Ltd., for example. This way, companies do not get subdivided in groups depending on their level of access authorizations on hardware.

One might argue that being able to install content from many sources on SEs would weaken their security, invoking for instance the hazards that content promiscuity can induce. However, interferences between applications, like the one demonstrated in this thesis about power consumption in Android, is not unavoidable. Good security policies and designs prevent them. Microkernels like seL4 [SeL], for example, are the proof that hosting data and executables from many third party sources can be done efficiently and securely.

Despite the current state of the industry, the Android Ready SE Alliance represents a spark of hope. By preprovisioning SEs with applications providing the most common needs, like cryptography based on usual algorithms, among others, some necessities required to make third party systems safe are covered. However, they might not be sufficient, and the deployment is moderate across the global fleet of Android products.

Today, fair competition in the domain of COTS personal devices is starting to be a major concern in the minds of regulatory institutions. In Europe, the Digital Markets Act (DMA) compels smartphones and tablets constructors to let users install third party applications, remove first party ones, and chose their favorite default ones. Apple has been the target of legal proceedings because of its reluctance to allow third party stores on iPhones [NS24], and to comply with the specification of Progressive Web Apps (PWAs) [Jav24]. It has also been sued by the United States Justice Department for similarly-minded practices, such as preventing third parties to access the hardware resources to implement payment services [MM24]. Both industrial and institutional actors have realized power balances are the crux of COTS devices exploitation.

5.2.3 Societal considerations

The advent of smartphones brought tighter restrictions on the rights of users toward their personal machines. Android, from its very beginnings, normalized having to do unconventional procedures to gain root access on one's phone, often requiring to inform the constructor about it. The apparition of some hardware security mechanisms, such as SEs, brought a second layer: the presence of, conceptually, small, independent platforms inside devices. These embed closed OSs and software, and users cannot manage them, nor can audit them. Beyond the technical aspects, this change of paradigm is important: now, constructors do not let users control their devices which are ironically made to run arbitrary software. This is not to say some services do not require guarantees: operating a payment necessarily imply to use verified, audited software and hardware. Nonetheless, it could be chosen by end users, and provided by third parties, similarly to how secure booting is organized on PCs. Instead, SEs were designed as if smartphones were Trojan horses around these enclaves relying on secrets and Non-Disclosure Agreements (NDAs). Google promised years ago to release the sources of Titan M, its hardware RoT. It never happened. However, the OpenTitan [low] initiative took place instead, dedicated to create a RoT chip with open design and firmware. As of right now, it is not ready for integration in mass produced COTS devices.

This relationship between multinational companies and users all around the world is founded on paternalism, where the former decide what is beneficial to the latter. Hence why some of these industrial actors, with economical and political weights similar to those of some countries, have strong enough public affairs departments to negotiate with international public organizations [CA23], raising state sovereignty concerns.

If it is assumed most users cannot handle all the complex security questions prevailing in their COTS devices, letting them choose who will be responsible for these is technically possible, whether it is a company, an association, or an individual. This role, unseen as of now, could be considered as a security planner. The natural or legal person in charge of it could be the one to decide, among other things:

- on what software is based the RoT. Today, the firmwares and OSs of SEs and TEEs are almost always secret, proprietary, and non-negotiable.
- which executables get to be installed in which trusted environments. As of now, these decisions are taken behind closed doors.
- which general security policies are enforced on the system. This includes SELinux rules, application-level ones, and more. A security planner could distinguish itself by, for instance, proposing strict regulation of fuel gauges accesses, contrary to the default one.

Various additional responsibilities linked to security updates and patches, along with countermeasures to discovered flaws could be included too. From a technical point of view, this delegation of security charges could be implemented with usual certificates: the constructors of processors and SEs would be the root CAs, and would sign the certificates of potential security planners. In the end, sensitive service providers and developers could select which security planners they trust.

This mode of organization, more respectful of end users, would likely not be developed across the industry of COTS products without regulation or the support of a major actor: this privilege distribution would challenge the hegemony of current holders, and accesses to devices' RoTs is a complete power struggle. But if applied, it would lead to an actual competition between varied actors with different forms, and the regulation of this market would be as straightforward as regulating the different online providers for a specific service. Transparency of security features could be a valuable argument for end users and third parties; as of now, it is not.

Manuscript and defense reports

Chapter content

Pascal Lafourcade's manuscript report 88
Clémentine Maurice's manuscript report 91
Thesis defense report 94

Pr. Pascal Lafourcade
Université Clermont Auvergne
Laboratoire d'Informatique Modélisation et d'Optimisation des Systèmes,
Campus des Cézeaux, 63170 Aubière, France
E-Mail: pascal.lafourcade@uca.fr

Le 17 juillet 2024, à Aubière,

Rapport sur le manuscrit de thèse de Vincent Giraud:
Sécurité des applications sur systèmes non maîtrisés. Étude des risques, protections, enjeux et intérêts autour de la confiance dans les produits informatiques sur étagère.

Dans son manuscrit de thèse, Vincent Giraud s'intéresse à la sécurité des applications. Cette problématique ne cesse de croître car les utilisateurs possèdent de plus en plus d'appareils connectés. Dans ce contexte, l'objectif de la thèse est d'analyser la sécurité de deux systèmes.

Le premier est un algorithme de chiffrement à clé publique bien connu : le chiffrement de McEliece. Ce chiffrement conçu en 1978 est un chiffrement à base de code correcteurs. Il s'agit d'une des solutions proposées pour résister à un ordinateur post-quantique. Dans ce cadre, Vincent propose une attaque par injection de faute sur ce chiffrement. En injectant une faute dans le calcul de déchiffrement, l'attaque proposée permet de retrouver la clé secrète utilisée.

Vincent propose ensuite de regarder la sécurité d'une application de saisie code PIN sur téléphone Android. Dans ce contexte, il montre comment utiliser les jauges de consommation énergétique pour monter une attaque par canaux auxiliaires. En analysant la consommation du téléphone, il est alors possible de trouver le code PIN saisi par l'utilisateur. Des résultats expérimentaux sur un téléphone Android montrent clairement comment monter cette attaque. Vincent propose également des contremesures simples pour éviter cette attaque.

Le manuscrit est clair et bien structuré. La thèse est constituée d'une synthèse en français, puis d'une introduction qui contient un chapitre sur le contexte et la motivation de la thèse et d'un autre chapitre présentant l'état de l'art. Ensuite, le manuscrit contient une partie sur les contributions qui est constituée de deux chapitres : un sur l'attaque par injection de fautes sur McEliece et un autre sur l'analyse de la sécurité d'un système de saisie de code PIN sur Android, avec une attaque par canaux auxiliaires. Enfin le manuscrit termine par une conclusion.

Introduction (Chapitre 1 et 2)

Dans ce premier chapitre, Vincent Giraud commence avec une présentation du contexte de la thèse. Il introduit la notion de produits informatiques sur étagère, la sécurité des plateformes matérielles,

les appareils embarqués et la sécurité reposant sur les logiciels.

Dans un second chapitre, Vincent Giraud fait les deux états de l'art correspondant à ses deux contributions. Il explique d'abord le fonctionnement de la cryptographie en boîte blanche. Approche qu'il va utiliser dans sa première contribution pour monter une attaque par injection de faute sur le chiffrement McEliece. En effet, il est important de connaître l'algorithme de chiffrement utilisé mais également son implémentation précise afin de pouvoir injecter la faute au bon endroit et au bon moment. Dans un second temps, Vincent introduit les mécanismes de sécurité utilisés sur Android. La connaissance des cet environnement va lui permettre dans sa seconde contribution de monter une attaque par canaux auxiliaires sur Android.

Contributions

Chapitre 3

Dans ce chapitre, qui constitue la contribution majeure de la thèse de Vincent Giraud, il commence par décrire l'algorithme de chiffrement et de déchiffrement de McEliece. Il explique que le cœur de l'algorithme de déchiffrement repose sur la multiplication d'un vecteur par un matrice de permutation secrète. L'idée de l'attaque est de modifier par une injection de faute une instruction de Ou-Exclusif (EOR) dans la boucle for de l'algorithme de multiplication pour la remplacer par une instruction de soustraction (RSB). Ce modèle d'attaque est original, il permet en une injection de faire en sorte que la faute soit exécutée à chaque tour de la boucle. En observant les résultats fautés du déchiffrement, il en déduit des contraintes sur les positions des 1 dans la matrice secrète de permutation. Cela permet de réduire considérablement l'espace possible des matrices de permutation. Il est alors possible de faire une recherche par brute force pour trouver la clé de déchiffrement. Vincent explique comment monter l'attaque en théorie mais n'a pas monté l'attaque en pratique. Cette attaque théorique est une première attaque par injection sur ce cryptosystème. Ce travail a donné lieu à deux dépôts de brevet, à une publication à *Security of Software/Hardware Interfaces (SILM) 2023, at European Symposium on Security and Privacy (EuroS&P)* et à une présentation à la Journée thématique sur les Attaques par Injection de Fautes (JAIF) 2022.

Chapitre 4

Dans ce chapitre, Vincent Giraud présente d'abord le fonctionnement des jauges de batteries sur Android (Battery Manager). Ces mécanismes sont clairement utiles pour l'utilisateur afin de mieux maîtriser l'utilisation de son appareil. L'idée est de détourner l'utilisation de ces mesures pour pouvoir déduire le code PIN de l'utilisateur lors de sa saisie. Vincent Giraud démontre pratiquement comment analyser ces données pour retrouver le code PIN de l'utilisateur saisi sur l'écran du téléphone sur un affichage. L'affichage des chiffres de 0 à 9 utilisé est l'affichage usuel et dans ce contexte est supposé connu. Ainsi, le principe est que chaque saisie d'un chiffre sur l'écran provoque une consommation électrique. Il suffit alors de regarder le temps entre deux saisies de chiffre pour déduire la distance entre les chiffres et donc de déduire leurs valeurs avec une grande certitude. Cette attaque a été réalisée sur un téléphone Android. Enfin Vincent propose des contre-mesures permettant d'éviter cette attaque, comme par exemple avoir un affichage aléatoire des chiffres de 0 à 9. Ce travail a donné lieu à un dépôt de brevet et à une publication internationale à *International Workshop on Security (IWSEC) 2023.*

Conclusion

Chapitre 5

Dans ce dernier chapitre de la thèse, Vincent Giraud rappelle l'ensemble des contributions de son manuscrit. Il montre aussi les impacts de ses deux contributions. Il propose également de nombreuses pistes de recherche prometteuses dans les différents domaines qu'il a étudiés.

Les travaux de Vincent touchent deux domaines de la sécurité : la cryptographie pour la première contribution sur le chiffrement McEliece et la sécurité appliquée sur Android dans la seconde contribution. Cette thèse présente deux contributions originales reposant sur l'injection de faute pour la première et les attaques par canaux auxiliaires pour la seconde. Les travaux sont à la fois théoriques et pratiques et ont fait l'objet de publications internationales et de trois dépôts de brevet. Pour toutes ces raisons, je recommande la soutenance de la thèse de Vincent Giraud.

Pascal LAFOURCADE

Dr. Clémentine Maurice
Chargée de Recherche CNRS
Centre Inria de l'Université de Lille
Parc scientifique de la Haute-Borne 40, avenue Halley - Bât B - Park Plaza
59650 Villeneuve d'Ascq – France
clementine.maurice@inria.fr
Tel. +33 (0)3 59 35 87 32

Lille, le 2 septembre 2024

Rapport de Clémentine MAURICE, Chargée de Recherche CNRS, laboratoire CRIStAL

sur le mémoire de thèse de doctorat présenté par

Vincent GIRAUD

en vue de l'obtention du grade de Docteur en Informatique

Le mémoire de thèse de Vincent GIRAUD, intitulé « Sécurité des applications sur systèmes non maîtrisés. Étude des risques, protections, enjeux et intérêts autour de la confiance dans les produits informatiques sur étagère », décrit les travaux de recherche qu'il a menés au sein de l'unité de recherche « DIENS - Département d'informatique de l'École normale supérieure », au sein de l'Ecole Doctorale Sciences Mathématiques de Paris Centre.

Les travaux réalisés par Vincent GIRAUD abordent une problématique essentielle dans le domaine de la sécurité des systèmes dits sur étagère. Ces systèmes sont complexes à sécuriser, car ils sont sous le contrôle des utilisateurs finaux, et les différents acteurs industriels ne peuvent pas nécessairement exploiter les mécanismes matériels de sécurité (qui ne sont pas accessibles à tous, ni présents sur tous les systèmes). Les travaux de recherche menés se concentrent sur le cryptosystème McEliece dans un contexte en boîte blanche, et une attaque par canal auxiliaire sur les entrées PIN des plateformes Android se basant sur la gestion de l'énergie. Le document écrit en anglais comprend 5 chapitres, dont les chapitres 3 et 4 portent sur les contributions originales du travail de doctorat.

Le chapitre 1 introduit le contexte et les motivations, et en particulier les mécanismes matériels de sécurité, tels que les modules de sécurité matériels (HSM), les modules de plateforme de confiance (TPM), les environnements d'exécution de confiance (TEE) ou les éléments sécurisés (SE), et la difficulté de leur utilisation pour des développeurs tiers.

Le chapitre 2 se concentre sur l'état de l'art concernant les deux contributions majeures de cette thèse : d'une part, les implémentations et les limites de la cryptographie en boîte blanche, et d'autre part, les différentes couches d'abstraction d'Android, leurs interactions, et la difficulté de protéger ce système. C'est un chapitre bien détaillé et agréable à lire car Vincent GIRAUD garde toujours une vision d'ensemble et un certain recul sur son sujet, en expliquant et motivant les différentes techniques et leurs approches historiques.

Le chapitre 3 étudie le cryptosystème McEliece sous deux angles : premièrement, le côté offensif, avec une nouvelle attaque par injection de faute sur une implémentation ARM qui permet à l'attaquant de récupérer la matrice de permutation (et donc d'en dériver la clé privée du système), et deuxièmement, le côté défensif, avec une proposition de variante de McEliece résistante à l'attaque montrée en première partie. L'attaque par injection de faute cible une instruction particulière avant le chargement du binaire, si bien que l'instruction est fautée tout le long de l'exécution. L'instruction ciblée est l'instruction `EOR` de la multiplication de matrice remplacée par l'instruction `RSB` (un seul bit change de valeur). L'attaque se base sur le fait que le poids de Hamming du résultat de la multiplication fautée du vecteur d'entrée par la matrice de permutation n'est pas nécessairement le même que celui du vecteur d'entrée. Avec des vecteurs d'entrée bien choisis, l'attaquant peut en déduire des ensembles de position de « 1 » de la matrice de permutation. Ces contraintes limitent grandement le nombre de matrices possibles, qui doivent ensuite être testées par force brute. Il n'est pas aisé de retranscrire les différentes étapes et l'idée générale d'un travail de cryptanalyse, mais Vincent GIRAUD a fait preuve de beaucoup de pédagogie dans ce chapitre, avec deux exemples (certes simplifiés) illustrant pas à pas les étapes de l'attaque après l'injection de faute. J'apprécie également que l'approche ne se limite pas au côté offensif, mais cherche également à proposer des corrections.

Le chapitre 4 porte sur la confidentialité inter-processus sur Android, et plus particulièrement sur un canal auxiliaire qui utilise l'indicateur de charge (*fuel gauge* en anglais) du système de gestion de l'alimentation. Le chapitre commence par une introduction sur la gestion de l'alimentation dans Android, du composant matériel à l'intégration logicielle et notamment la politique d'accès d'Android aux différents capteurs et au gestionnaire de la batterie. Vincent GIRAUD développe ensuite une attaque qui espionne un code PIN tapé par l'utilisateur grâce au gestionnaire de batterie, qui rend des informations très fines sur le courant instantané. L'idée de l'attaque repose sur le fait que toucher l'écran pour appuyer sur un chiffre provoque un pic de consommation de courant, qui se voit donc à travers l'API du gestionnaire de batterie. L'attaquant obtient donc une suite temporelle de pics, et peut déduire la distance physique des chiffres tapés sur le clavier virtuel en analysant le temps entre deux pics (par exemple, les chiffres 1 et 2 sont proches physiquement, tandis que 9 est plus éloigné de 1, les pics entre 1 et 2 seront donc plus rapprochés qu'entre 1 et 9). Je regrette l'absence de détail quant aux expérimentations qui ont été menées : combien de traces ont été récoltées, sur combien de codes PIN ? Nous avons une trace du code 1473, mais il est difficile de voir comment l'évaluation se généralise au vu de la présentation des résultats. Par exemple : « When good measures have been acquired, it is very frequent to get less than 10 propositions, or even less than 5 sometimes, for PIN codes containing 4 numbers. », le lecteur se demande qu'est-ce qu'une bonne mesure, dans quelles proportions l'attaque rend-elle 10 ou 5 propositions, et à quelle position se situe le bon code PIN dans ces propositions. Il aurait également été intéressant de comparer de façon plus approfondie, au moins de façon conceptuelle, les solutions développées et les approches existantes dans l'état de l'art (notamment les papiers cités TouchLogger et TapLogger qui ont comme but similaire de retrouver les entrées utilisateur à l'aide des capteurs du téléphone). Cela aurait permis de mieux mettre en perspective les contributions développées.

Le chapitre 5 conclut le manuscrit avec d'une part un rappel des contributions et leur impact, et d'autre part, des pistes de réflexion quant à la poursuite de ces travaux, ainsi que des considérations industrielles et sociétales. Ces deux dernières parties sont intéressantes car elles prennent du recul sur le marché des constructeurs, et notamment la place des éléments sécurisés dans nos appareils au quotidien, ainsi que sur les choix que peuvent exercer les utilisateurs finaux sur leurs propres machines.

En conclusion, dans cette thèse, Vincent GIRAUD a exploré deux facettes bien différentes l'une de l'autre de la sécurité des systèmes sur étagère, avec d'un côté une attaque en cryptanalyse qui requiert des compétences solides en cryptographie, et de l'autre une attaque qui se base beaucoup plus sur la connaissance du matériel et du système Android. Ce sont deux approches qui se rejoignent rarement dans

une thèse, mais qui sont complémentaires et témoignent de la capacité de Vincent GIRAUD à appréhender son sujet de manière globale. Le manuscrit lui-même est bien écrit, avec beaucoup de contexte et de pédagogie. Ces travaux ont bien été valorisés, car ils ont été publiés dans deux workshops internationaux (SILM et IWSEC), ont fait l'objet de communications nationales (JAIF et SSTIC) et également de deux demandes de brevets auprès de l'INPI.

Au vu de l'ensemble de ces éléments, je donne un avis favorable pour que Vincent GIRAUD présente ses travaux de recherche lors d'une soutenance de thèse.

Clémentine Maurice

DOCTORAT de l'Université PSL

RAPPORT DE SOUTENANCE

Nom et prénom du doctorant : GIRAUD Vincent

Date de la soutenance : 26 septembre 2024

Président du Jury : LAFOURCADE Pascal

Dans sa thèse, Vincent Giraud étudie la sécurité des applications dans les produits informatiques sur étagère. Il a fait une présentation détaillée, et bien structurée des travaux de sa thèse. Au cours de sa présentation claire et pédagogique, il a d'abord introduit le contexte de ses travaux, en particulier la cryptographie en boîte blanche et les mécanismes de sécurité sur Android. Il a ensuite présenté les deux contributions de sa thèse. Il a d'abord expliqué le chiffrement de McEliece publié en 1978, chiffrement à clé publique basé sur les codes correcteurs pour lequel il a montré une nouvelle attaque théorique par injection de faute. Cette approche vise à retrouver la matrice de permutation. Il s'agit de la première attaque par injection de faute sur ce cryptosystème. Ensuite, il a abordé la sécurité des jauges de carburant sur Android. Il a clairement expliqué que ces jauges permettent de monter une attaque par « side channels » (canaux auxiliaires) pour deviner le code PIN saisi par un utilisateur. Il a également réalisé un prototype pour montrer la faisabilité de cette attaque sur Android.

La présentation confirme sa compréhension de son sujet et sa capacité à monter des attaques aussi bien théoriques que pratiques. Il a également été en mesure de proposer des perspectives à ses travaux de recherche. Vincent a fait preuve d'une grande maturité et a réussi à donner des réponses pertinentes aux diverses questions des membres du jury.

Dans l'ensemble, le jury a conclu à l'unanimité que Vincent Giraud que son travail peut avoir un impact important en sécurité appliquée aussi bien d'un point de vue théorique que pratique. Il a démontré toutes les capacités pour devenir chercheur ou enseignant-chercheur de par le large spectre des techniques qu'il a mises en place mais aussi le contexte interdisciplinaire de ses recherches. Le jury a donc décidé de décerner à Vincent Giraud le titre de Docteur en informatique de l'Université PSL.

Prénom et Nom	Signature	Prénom et Nom	Signature
David NACCACHE	Visio	Pascal LAFOURCADE	
Clémentine MAURICE		Aurélien FRANCILLON	Visio
Sophie QUINTON		Guillaume BOUFFARD	

RÉPUBLIQUE FRANÇAISE
ENS
ÉCOLE NORMALE SUPÉRIEURE

Scientific communications

Academic

- *Journée thématique sur les Attaques par Injection de Fautes* (JAIF) 2022: *L'attaque en faute : la bête noire des boîtes blanches.*

- *Security of Software/Hardware Interfaces* (SILM) 2023, at *European Symposium on Security and Privacy* (EuroS&P): *Faulting original McEliece's implementations is possible — How to mitigate this risk ?*

- *Symposium sur la Sécurité des Technologies de l'Information et des Communications* (SSTIC) 2023: *Batterie à bord, quand les jauges de carburant dépassent les limites.*

- *International Workshop on Security* (IWSEC) 2023: *Power analysis pushed too far — Breaking Android-based isolation with fuel gauges.* Associated publication in *Lecture Notes in Computer Science* (LNCS) volume 14128.

Industrial

- Patent application number WO2024083849 filed at the *Institut National de la Propriété Industrielle* (INPI) on October 17, 2022: *Encodage en boîte blanche.*

- Patent application number WO2024083855 filed at the *Institut National de la Propriété Industrielle* (INPI) on October 17, 2022: *Clés cryptographiques en boîte blanche.*

- Patent application number WO2024121142 filed at the *Institut National de la Propriété Industrielle* (INPI) on December 6, 2022: *Procédé de protection d'un terminal contre une attaque par canal auxiliaire.*

Bibliography

[ABC$^+$22] Albrecht, M.R., Bernstein, D.J., Chou, T., Cid, C., Gilcher, J., Lange, T., Maram, V., Maurich, I.v., Misoczki, R., Niederhagen, R., Paterson, K.G., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., Tjhai, C.J., Tomlinson, M., and Wang, W. Classic McEliece: Conservative code-based cryptography: Cryptosystem specification, October 2022. URL: https://classic.mceliece.org/mceliece-spec-20221023.pdf. 32

[ACG15] Will Arthur, David Challener, and Kenneth Goldman. History of the TPM. In Will Arthur, David Challener, and Kenneth Goldman, editors, *A Practical Guide to TPM 2.0: Using the New Trusted Platform Module in the New Age of Security*, pages 1–5. Apress, Berkeley, CA, 2015. doi:10.1007/978-1-4302-6584-9_1. 8

[AF04] Tiago Alves and Don Felton. Trustzone : Integrated hardware and software security. *Information Quarterly*, 3:18–24, 2004. 8

[AHPT10] R. M. Avanzi, S. Hoerder, D. Page, and M. Tunstall. Side-Channel Attacks on the McEliece and Niederreiter Public-Key Cryptosystems, 2010. URL: https://eprint.iacr.org/2010/479. 33

[AKPK16] Elias Athanasopoulos, Vasileios P. Kemerlis, Georgios Portokalidis, and Angelos D. Keromytis. NaClDroid: Native Code Isolation for Android Applications. In Ioannis Askoxylakis, Sotiris Ioannidis, Sokratis Katsikas, and Catherine Meadows, editors, *Computer Security – ESORICS 2016*, Lecture Notes in Computer Science, pages 422–439, Cham, 2016. Springer International Publishing. doi:10.1007/978-3-319-45744-4_21. 24

[ANS] ANSSI. iTrustee Version 5.0 certification. URL: https://cyber.gouv.fr/produits-certifies/itrustee-version-50. 9

[ANS22] ANSSI views on the Post-Quantum Cryptography transition, 2022. URL: https://www.ssi.gouv.fr/publication/anssi-views-on-the-post-quantum-cryptography-transition/. 32

[App] Apple Pay home page. URL: https://www.apple.com/apple-pay/. 11

[App22] Apple. Apple Platform Security Guide. 2022. URL: https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf. 10, 11

[Ave17] Guillaume Averlant. Multi-level Isolation for Android Applications. In *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 128–131, October 2017. URL: https://ieeexplore.ieee.org/abstract/document/8109273, doi:10.1109/ISSREW.2017.61. 24

[Bar20] Lucas Barthelemy. Toward an Asymmetric White-Box Proposal. Technical Report 893, 2020. URL: https://eprint.iacr.org/2020/893. 21

[BBD+22] Guillaume Barbu, Ward Beullens, Emmanuelle Dottax, Christophe Giraud, Agathe Houzelot, Chaoyun Li, Mohammad Mahzoun, Adrián Ranea, and Jianrui Xie. ECDSA White-Box Implementations: Attacks and Designs from WhibOx 2021 Contest, 2022. URL: https://eprint.iacr.org/2022/385. xiv, 22

[BBMT18] Estuardo Alpirez Bock, Chris Brzuska, Wil Michiels, and Alexander Treff. On the Ineffectiveness of Internal Encodings - Revisiting the DCA Attack on White-Box Cryptography, 2018. URL: https://eprint.iacr.org/2018/301. 21

[BCD+18] Andrea Biondo, Mauro Conti, Lucas Davi, Tommaso Frassetto, and Ahmad-Reza Sadeghi. The Guard's Dilemma: Efficient {Code-Reuse} Attacks Against Intel {SGX}. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1213–1227, 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/biondo. 8

[BCGO09] Thierry P. Berger, Pierre-Louis Cayrel, Philippe Gaborit, and Ayoub Otmani. Reducing Key Length of the McEliece Cryptosystem. In Bart Preneel, editor, *Progress in Cryptology – AFRICACRYPT 2009*, Lecture Notes in Computer Science, pages 77–97, Berlin, Heidelberg, 2009. Springer. doi:10.1007/978-3-642-02384-2_6. 33

[BDD+11] Sven Bugiel, Lucas Davi, Alexandra Dmitrienko, Stephan Heuser, Ahmad-Reza Sadeghi, and Bhargava Shastry. Practical and lightweight domain isolation on Android. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, pages 51–62, New York, NY, USA, October 2011. Association for Computing Machinery. doi:10.1145/2046614.2046624. 24

[BDG+22] Sven Bauer, Hermann Drexler, Maximilian Gebhardt, Dominik Klein, Friederike Laus, and Johannes Mittmann. Attacks Against White-Box ECDSA and Discussion of Countermeasures - A Report on the WhibOx Contest 2021, 2022. URL: https://eprint.iacr.org/2022/448. xiv, 22

[Ber10] Daniel J. Bernstein. Grover vs. McEliece. In Nicolas Sendrier, editor, *Post-Quantum Cryptography*, Lecture Notes in Computer Science, pages 73–80, Berlin, Heidelberg, 2010. Springer. doi:10.1007/978-3-642-12929-2_6. 33

[Ber11] Daniel J. Bernstein. List Decoding for Binary Goppa Codes. In Yeow Meng Chee, Zhenbo Guo, San Ling, Fengjing Shao, Yuansheng Tang, Huaxiong Wang, and Chaoping Xing, editors, *Coding and Cryptology*, Lecture Notes in Computer Science, pages 62–80, Berlin, Heidelberg, 2011. Springer. doi:10.1007/978-3-642-20901-7_4. 43

[BGE05] Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a White Box AES Implementation. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography*, Lecture Notes in Computer Science, pages 227–240, Berlin, Heidelberg, 2005. Springer. doi:10.1007/978-3-540-30564-4_16. 20

[BGG21] Guillaume Bouffard, Vincent Giraud, and Léo Gaspard. Java Card Virtual Machine Memory Organization: A Design Proposal, October 2021. URL: http://arxiv.org/abs/2110.10037, arXiv:2110.10037, doi:10.48550/arXiv.2110.10037. 4

[BHMT15] Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. Differential Computation Analysis: Hiding your White-Box Designs is Not Enough, 2015. URL: https://eprint.iacr.org/2015/753. 21

[BJB17] David Berend, Bernhard Jungk, and Shivam Bhasin. There Goes Your PIN: Exploiting Smartphone Sensor Fusion Under Single and Cross User Setting. *Cryptology ePrint Archive*, 2017. URL: https://eprint.iacr.org/2017/1169. 26

[BKE+17] Ben Gras, Kaveh Razavi, Erik Bosman, Herbert Box, and Cristiano Giuffrida. ASLR on the Line: Practical Cache Attacks on the MMU. In *NDSS Symposium*, 2017. URL: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/aslrcache-practical-cache-attacks-mmu/. 22

[BKL+13] Guillaume Bouffard, Tom Khefif, Jean-Louis Lanet, Ismael Kane, and Sergio Casanova Salvia. Accessing secure information using export file fraudulence. In *2013 International Conference on Risks and Security of Internet and Systems (CRiSIS)*, pages 1–5, October 2013. URL: https://ieeexplore.ieee.org/document/6766346, doi:10.1109/CRiSIS.2013.6766346. 20

[BKS+22] Pietro Borrello, Andreas Kogler, Martin Schwarzl, Moritz Lipp, Daniel Gruss, and Michael Schwarz. {ÆPIC} Leak: Architecturally Leaking Uninitialized Data from the Microarchitecture. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3917–3934, 2022. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/borrello. 8

[BLLL15] Guillaume Bouffard, Michael Lackner, Jean-Louis Lanet, and Johannes Loinig. Heap... Hop! Heap Is Also Vulnerable. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications*, Lecture Notes in Computer Science, pages 18–31, Cham, 2015. Springer International Publishing. doi:10.1007/978-3-319-16763-3_2. 20

[BLP08] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Attacking and Defending the McEliece Cryptosystem. In Johannes Buchmann and Jintai Ding, editors, *Post-Quantum Cryptography*, Lecture Notes in Computer Science, pages 31–46, Berlin, Heidelberg, 2008. Springer. doi:10.1007/978-3-540-88403-3_3. 32

[BLP11] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Wild McEliece. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography*, Lecture Notes in Computer Science, pages 143–158, Berlin, Heidelberg, 2011. Springer. doi:10.1007/978-3-642-19574-7_10. 33

[BMH78] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems (Corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, May 1978. doi:10.1109/TIT.1978.1055873. 31

[BMW+18] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the Keys to the Intel {SGX} Kingdom with Transient {Out-of-Order} Execution. In *27th USENIX Security Symposium (USENIX Security 18)*, page 991, 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/bulck. 8

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, Lecture Notes in Computer Science, pages 513–525, Berlin, Heidelberg, 1997. Springer. doi:10.1007/BFb0052259. 20

[BT20] Estuardo Alpirez Bock and Alexander Treff. Security Assessment of White-Box Design Submissions of the CHES 2017 CTF Challenge, 2020. URL: https://eprint.iacr.org/2020/342. 20

[Bun20] Bundesamt für Sicherheit in der Informationstechnik. Migration zu Post-Quanten-Kryptografie - Handlungsempfehlungen des BSI. August 2020. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Krypto/Post-Quanten-Kryptografie.pdf. 32

[BWK+17] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. Telling Your Secrets without Page Faults: Stealthy Page {Table-Based} Attacks on Enclaved Execution. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1041–1056, 2017. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/van-bulck. 8

[BXZ+17] Mahmoud A. Bokhari, Yuanzhong Xia, Bo Zhou, Brad Alexander, and Markus Wagner. Validation of Internal Meters of Mobile Android Devices, January 2017. URL: http://arxiv.org/abs/1701.07095, arXiv:1701.07095, doi:10.48550/arXiv.1701.07095. xviii, 60

[BYW17] Xiaolong Bai, Jie Yin, and Yu-Ping Wang. Sensor Guardian: Prevent privacy inference on Android sensors. *EURASIP Journal on Information Security*, 2017(1):10, June 2017. doi:10.1186/s13635-017-0061-8. 26

[CA23] Catherine Kessedjian and Anne-Thida Norodom. Qui fabrique le droit international ? *Le Monde diplomatique*, page 22, May 2023. URL: https://www.monde-diplomatique.fr/2023/05/KESSEDJIAN/65745. 85

[Car] Carte Vitale application home page. URL: https://www.applicartevitale.fr/. 5

[CB19] Andrea Caforio and Subhadeep Banik. A Study of Persistent Fault Analysis. In Shivam Bhasin, Avi Mendelson, and Mridul Nandi, editors, *Security, Privacy, and Applied Cryptography Engineering*, Lecture Notes in Computer Science, pages 13–33, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-35869-3_4. 20

[CC98] A. Canteaut and F. Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, January 1998. doi:10.1109/18.651067. 31

[CC11] Liang Cai and Hao Chen. TouchLogger: Inferring Keystrokes On Touch Screen From Smartphone Motion. page 6, 2011. 26

[CCBG19] Matteo Cardaioli, Mauro Conti, Kiran Balagani, and Paolo Gasti. Your PIN Sounds Good! On The Feasibility of PIN Inference Through Audio Leakage, May 2019. URL: http://arxiv.org/abs/1905.08742, arXiv:1905.08742, doi:10.48550/arXiv.1905.08742. 26, 72

[CCD+20] Pierre-Louis Cayrel, Brice Colombier, Vlad-Florin Dragoi, Alexandre Menu, and Lilian Bossuet. Message-recovery Laser Fault Injection Attack on the Classic McEliece Cryptosystem, 2020. URL: https://eprint.iacr.org/2020/900. 33

[CCX+19] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. SgxPectre Attacks: Stealing Intel Secrets from SGX Enclaves via Speculative Execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 142–157, June 2019. URL: http://arxiv.org/abs/1802.09085, arXiv:1802.09085, doi:10.1109/EuroSP.2019.00020. 8

[CD10] Pierre-Louis Cayrel and Pierre Dusart. McEliece/Niederreiter PKC: Sensitivity to Fault Injection. In *2010 5th International Conference on Future Information Technology*, pages 1–6, May 2010. doi:10.1109/FUTURETECH.2010.5482663. 33

[CEJP03a] Stanley Chow, Phil Eisen, Harold Johnson, and Paul C. van Oorschot. A White-Box DES Implementation for DRM Applications. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, and Joan Feigenbaum, editors, *Digital Rights Management*, volume 2696, pages 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. URL: http://link.springer.com/10.1007/978-3-540-44993-5_1, doi:10.1007/978-3-540-44993-5_1. xiii, 16, 80

[CEJP03b] Stanley Chow, Philip Eisen, Harold Johnson, and Paul C. van Oorschot. White-Box Cryptography and an AES Implementation. In Kaisa Nyberg and Howard Heys, editors, *Selected Areas in Cryptography*, volume 2595, pages 250–270. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. URL: http://link.springer.com/10.1007/3-540-36492-7_17, doi:10.1007/3-540-36492-7_17. xiii, 16, 52, 80

[CEVMS14] Cong Chen, Thomas Eisenbarth, Ingo Von Maurich, and Rainer Steinwandt. Differential Power Analysis of a McEliece Cryptosystem, 2014. URL: https://eprint.iacr.org/2014/534. 33

[CFS01] Nicolas T. Courtois, Matthieu Finiasz, and Nicolas Sendrier. How to Achieve a McEliece-Based Digital Signature Scheme. In Colin Boyd, editor, *Advances in Cryptology — ASIACRYPT 2001*, Lecture Notes in Computer Science, pages 157–174, Berlin, Heidelberg, 2001. Springer. doi:10.1007/3-540-45682-1_10. xv, 32

[CGYW21] Patrick Cronin, Xing Gao, Chengmo Yang, and Haining Wang. Charger-Surfing: Exploiting a Power Line Side-Channel for Smartphone Information Leakage. 2021. 26

[Cho] Amit Chowdhry. Uber: Users Are More Likely To Pay Surge Pricing If Their Phone Battery Is Low. URL: https://www.forbes.com/sites/amitchowdhry/2016/05/25/uber-low-battery/. 66

[CMP14] Alain Couvreur, Irene Márquez-Corbella, and Ruud Pellikaan. A polynomial time attack against algebraic geometry code based public key cryptosystems. In *2014 IEEE International Symposium on Information Theory*, pages 1446–1450, June 2014. URL: https://ieeexplore.ieee.org/abstract/document/6875072, doi:10.1109/ISIT.2014.6875072. 33

[CMTE22] Colin Chibaya, Mfundo Monchwe, Taryn Nicole Michael, and Eli Bila Nimy. Showcasing White-Box Implementation of the RSA Digital Signature Scheme. *American Journal of Computer Science and Technology*, 2022. URL: https://www.sciencepublishinggroup.com/article/10072582. 21

[COT17] Alain Couvreur, Ayoub Otmani, and Jean-Pierre Tillich. Polynomial Time Attack on Wild McEliece Over Quadratic Extensions. *IEEE Transactions on Information Theory*, 63(1):404–427, January 2017. URL: https://ieeexplore.ieee.org/abstract/document/7496988, doi:10.1109/TIT.2016.2574841. 33

[CP16] Charles Hubain and Philippe Teuwen. Design de cryptographie white-box : et à la fin, c'est Kerckhoffs qui gagne. In *SSTIC 2016*, Rennes, 2016. URL: https://www.sstic.org/2016/presentation/design_de_cryptographie_white-box_et_a_la_fin_c_est_kerckhoffs_qui_gagne/. xiv, 6

[CPM+18] 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*, CCS '18, pages 163–177, New York, NY, USA, October 2018. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/3243734.3243802, doi:10.1145/3243734.3243802. 20

[CS98] Anne Canteaut and Nicolas Sendrier. Cryptanalysis of the Original McEliece Cryptosystem. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology — ASIACRYPT'98*, Lecture Notes in Computer Science, pages 187–199, Berlin, Heidelberg, 1998. Springer. doi:10.1007/3-540-49649-1_16. 31

[CS13] Stephen Checkoway and Hovav Shacham. Iago attacks: Why the system call API is a bad untrusted RPC interface. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 253–264, New York, NY, USA, March 2013. Association for Computing Machinery. doi:10.1145/2451116.2451145. 8

[CTu22] CTurt. Mast1c0re: Hacking the PS4 / PS5 through the PS2 Emulator - Part 1 - Escape, 2022. URL: https://cturt.github.io/mast1c0re.html. 23

[CVM+21] 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 Security 21)*, pages 699–716, 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/chen-zitai. 8

[CYS+21] Jinhua Cui, Jason Zhijingcheng Yu, Shweta Shinde, Prateek Saxena, and Zhiping Cai. SmashEx: Smashing SGX Enclaves Using Exceptions. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pages 779–793, New York, NY, USA, November 2021. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/3460120.3484821, doi:10.1145/3460120.3484821. 8

[DGH21] Emmanuelle Dottax, Christophe Giraud, and Agathe Houzelot. White-Box ECDSA: Challenges and Existing Solutions. In Shivam Bhasin and Fabrizio De Santis, editors, *Constructive Side-Channel Analysis and Secure Design*, Lecture Notes in Computer Science, pages 184–201, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-89915-8_9. 22

[DK20] Julian Danner and Martin Kreuzer. A fault attack on the Niederreiter cryptosystem using binary irreducible Goppa codes. *journal of Groups, complexity, cryptology*, Volume 12, Issue 1:6074, March 2020. URL: http://arxiv.org/abs/2002.01455, arXiv:2002.01455, doi:10.46298/jgcc.2020.12.1.6074. 33

[DLPR13] Cécile Delerablée, Tancrède Lepoint, Pascal Paillier, and Matthieu Rivain. White-Box Security Notions for Symmetric Encryption Schemes. Technical Report 523, 2013. URL: https://eprint.iacr.org/2013/523. xiv, 19

[DLV03a] P. Dusart, G. Letourneux, and O. Vivolo. Differential Fault Analysis on A.E.S., 2003. URL: https://eprint.iacr.org/2003/010. 21

[DLV03b] Pierre Dusart, Gilles Letourneux, and Olivier Vivolo. Differential Fault Analysis on A.E.S. In Jianying Zhou, Moti Yung, and Yongfei Han, editors, *Applied Cryptography and Network Security*, Lecture Notes in Computer Science, pages 293–306, Berlin, Heidelberg, 2003. Springer. doi:10.1007/978-3-540-45203-4_23. 20

[DMRP13] Yoni De Mulder, Peter Roelse, and Bart Preneel. Cryptanalysis of the Xiao – Lai White-Box AES Implementation. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography*, Lecture Notes in Computer Science, pages 34–49, Berlin, Heidelberg, 2013. Springer. doi:10.1007/978-3-642-35999-6_3. 20

[DMVN15] Batsayan Das, Lakshmipadmaja Maddali, and Harshita Vani Nallagonda. Role-based privilege isolation: A novel authorization model for Android smart devices. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 220–225, December 2015. URL: https://ieeexplore.ieee.org/abstract/document/7412093, doi:10.1109/ICITST.2015.7412093. 24

[DMWP10] Yoni De Mulder, Brecht Wyseur, and Bart Preneel. Cryptanalysis of a Perturbated White-Box AES Implementation. In Guang Gong and Kishan Chand Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010*, Lecture Notes in Computer Science, pages 292–310, Berlin, Heidelberg, 2010. Springer. doi:10.1007/978-3-642-17401-8_21. 20

[DWTN15] Daniel Augot, Lejla Batina, Daniel J. Bernstein, Joppe Bos, Johannes Buchmann,, Wouter Castryck, Orr Dunkelman, Tim Güneysu, Shay Gueron, Andreas Hülsing,, Tanja Lange, Mohamed Saied Emam Mohamed, Christian Rechberger, Peter Schwabe,, and Nicolas Sendrier, Frederik Vercauteren, Bo-Yin Yang. Post-Quantum Cryptography for Long-Term Security, 2015. URL: https://pqcrypto.eu.org/docs/initial-recommendations.pdf. xv, 32, 49

[ECJ15] Eloi Sanfelix, Cristofaro Mune, and Job de Haas. Practical attacks against Obfuscated Ciphers. page 38, 2015. 21

[EFK16] Alaa Mahmoud Elsobky, Abdelalim Kamal Farag, and Arabi Keshk. Efficient Implementation of McEliece Cryptosystem on Graphic Processing Unit. In *Proceedings of the 10th International Conference on Informatics and Systems*, INFOS '16, pages 247–253, New York, NY, USA, May 2016. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/2908446.2908491, doi:10.1145/2908446.2908491. 32

[ERAP18] Dmitry Evtyushkin, Ryan Riley, Nael CSE and ECE Abu-Ghazaleh, and Dmitry Ponomarev. BranchScope: A New Side-Channel Attack on Directional Branch Predictor. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 693–707, New York, NY, USA, March 2018. Association for Computing Machinery. doi:10.1145/3173162.3173204. 8

[Eur23] European Commission. The European Digital Identity Wallet Architecture and Reference Framework, February 2023. URL: https://digital-strategy.ec.europa.eu/en/library/european-digital-identity-wallet-architecture-and-reference-framework. 5

[FHW+20] Qi Feng, Debiao He, Huaqun Wang, Neeraj Kumar, and Kim-Kwang Raymond Choo. White-Box Implementation of Shamir's Identity-Based Signature Scheme. *IEEE Systems Journal*, 14(2):1820–1829, June 2020. doi:10.1109/JSYST.2019.2910934. 21

[FKK10] Denis Foo Kune and Yongdae Kim. Timing attacks on PIN input devices. In *Proceedings of the 17th ACM Conference on Computer and Communications Secu-*

rity, CCS '10, pages 678–680, New York, NY, USA, October 2010. Association for Computing Machinery. doi:10.1145/1866307.1866395. 26, 71

[FM08] Cédric Faure and Lorenz Minder. Cryptanalysis of the McEliece cryptosystem over hyperelliptic codes. *Eleventh International Workshop on Algebraic and Combinatorial Coding Theory*, June 2008. 33

[FOPT10] Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. Algebraic Cryptanalysis of McEliece Variants with Compact Keys. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, Lecture Notes in Computer Science, pages 279–298, Berlin, Heidelberg, 2010. Springer. doi:10.1007/978-3-642-13190-5_14. 33

[FPP23] Florent Valette, Patrice Hameau, and Philippe Thierry. From dusk till dawn: toward an effective trusted UI. In *SSTIC 2023*, Rennes, 2023. URL: https://www.sstic.org/2023/presentation/from_dusk_till_dawn_toward_an_effective_trusted_ui/. xix, 76

[Fra] France Identité home page. URL: https://france-identite.gouv.fr/. 5

[Fre] FreeRTOS home page. URL: https://freertos.org/. 4

[FXX+18] Fan Zhang, Xiaoxuan Lou, Xinjie Zhao, Shivam Bhasin, Wei He, Ruyi Ding, Samiya Qureshi, and Kui Ren. Persistent Fault Analysis on Block Ciphers | IACR Transactions on Cryptographic Hardware and Embedded Systems. In *TCHES*. IACR, 2018. URL: https://tches.iacr.org/index.php/TCHES/article/view/7272. 20

[Gab05] Philippe Gaborit. Shorter keys for code-based cryptography. *Proceedings of the 2005 International Workshop on Coding and Cryptography (WCC 2005)*, January 2005. 33

[GG22] Pierre Galissant and Louis Goubin. Resisting Key-Extraction and Code-Compression: A Secure Implementation of the HFE Signature Scheme in the White-Box Model. Technical Report 138, 2022. URL: https://eprint.iacr.org/2022/138. 21

[GH18] Robert Gawlik and Thorsten Holz. {SoK}: Make {JIT-Spray} Great Again. In *12th USENIX Workshop on Offensive Technologies (WOOT 18)*, 2018. URL: https://www.usenix.org/conference/woot18/presentation/gawlik. 23

[GIE] GIE CB. CB en chiffres. URL: https://www.cartes-bancaires.com/cb/chiffres/. 13

[Git20] Louison Gitzinger. *Surviving the Massive Proliferation of Mobile Malware*. These de doctorat, Rennes 1, December 2020. URL: https://www.theses.fr/fr/2020REN1S058. 24

[Glo10] GlobalPlatform. TEE Client API Specification v1.0, GPD_SPE_007, 2010. URL: https://globalplatform.org/specs-library/tee-client-api-specification/. 8

[GMQ07] Louis Goubin, Jean-Michel Masereel, and Michaël Quisquater. Cryptanalysis of White Box DES Implementations. In Carlisle Adams, Ali Miri, and Michael Wiener, editors, *Selected Areas in Cryptography*, Lecture Notes in Computer Science, pages 278–295, Berlin, Heidelberg, 2007. Springer. doi:10.1007/978-3-540-77360-3_18. 20

[Gooa] Google Pay home page. URL: https://pay.google.com/intl/en_us/about/. 11

[Goob] Google Inc. Trusty TEE. URL: https://source.android.com/docs/security/features/trusty. 9

[Goo15] Google Inc. Android 6.0 Compatibility Definition Document (CDD), 2015. URL: https://source.android.google.cn/static/docs/compatibility/6.0/android-6.0-cdd.pdf. 9

[Goo23] Google Inc. Android 14 Compatibility Definition Document (CDD), 2023. URL: https://source.android.google.cn/docs/compatibility/14/android-14-cdd. 9

[Gra] GrapheneOS home page. URL: https://grapheneos.org/. 74

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219, New York, NY, USA, July 1996. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/237814.237866, doi:10.1145/237814.237866. 33

[Gro97] Lov K. Grover. Quantum Mechanics helps in searching for a needle in a haystack. *Physical Review Letters*, 79(2):325–328, July 1997. URL: http://arxiv.org/abs/quant-ph/9706033, arXiv:quant-ph/9706033, doi:10.1103/PhysRevLett.79.325. 33

[GSY14] Manu Gulati, Michael J. Smith, and Shu-Yi Yu. Security enclave processor for a system on a chip, September 2014. URL: https://patents.google.com/patent/US8832465/en. 10

[Har86] Niederreiter Harald. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15(2):157–166, 1986. 33

[HMP10] Stefan Heyse, Amir Moradi, and Christof Paar. Practical Power Analysis Attacks on Software Implementations of McEliece. In Nicolas Sendrier, editor, *Post-Quantum Cryptography*, Lecture Notes in Computer Science, pages 108–125, Berlin, Heidelberg, 2010. Springer. doi:10.1007/978-3-642-12929-2_9. 33

[HTG+11] Helmut Hlavacs, Thomas Treutner, Jean-Patrick Gelas, Laurent Lefevre, and Anne-Cecile Orgerie. Energy Consumption Side-Channel Attack at Virtual Machines in a Cloud. In *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, pages 605–612, December 2011. URL: https://ieeexplore.ieee.org/document/6119058, doi:10.1109/DASC.2011.110. 20

[IEE13] IEEE. IEEE Standard for Identity-Based Cryptographic Techniques using Pairings. Technical report, IEEE, 2013. URL: http://ieeexplore.ieee.org/document/6662370/, doi:10.1109/IEEESTD.2013.6662370. 21

[iFi] iFixit. Nintendo Switch (2017 HAC-001) Motherboard: Replacement Part. URL: https://www.ifixit.com/products/nintendo-switch-2017-hac-001-motherboard. 61

[Inf22] Information Technology Laboratory, Computer Security Division. Announcing PQC Candidates to be Standardized, Plus Fourth Round Candidates, March 2022. URL: https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4. xv, 32

[Int] Intel. Trust Domain Extensions home page. URL: https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/overview.html. 8

[ISO11] ISO/IEC JTC 1/SC 17. ISO/IEC 7816, 2011. 16

[ISO15] ISO/IEC JTC 1. ISO/IEC 11889, 2015. 7

[Jap21] Japan Electronics and Information Technology Industries Association (JEITA). A Guide to the Safe Use of Lithium-Ion Secondary Cells in Notebook-type Personal Computers and Tablet Terminal, February 2021. URL: https://home.jeita.or.jp/upload_file/20210407112651_JbDgUSWGiB.pdf. 59

[Jav24] Javier Espinoza. EU seeks to investigate Apple over cutting off web apps. *Financial Times*, February 2024. URL: https://www.ft.com/content/d2f7328c-5851-4f16-8f8d-93f0098b6adc. 84

[JBF03] Matthias Jacob, Dan Boneh, and Edward Felten. Attacking an Obfuscated Cipher by Injecting Faults. In Joan Feigenbaum, editor, *Digital Rights Management*, Lecture Notes in Computer Science, pages 16–31, Berlin, Heidelberg, 2003. Springer. doi:10.1007/978-3-540-44993-5_2. 21

[Jer13] Jeremiah Cox. Microsoft UEFI Certification Authority. In *UEFI PlugFest*, September 2013. URL: https://uefi.org/sites/default/files/resources/UEFI_Plugfest_2013_-_New_Orleans_-_Microsoft_UEFI_CA.PDF. 84

[JIT24] JIT Support on iOS · Issue #3 · mozilla/platform-tilt, 2024. URL: https://github.com/mozilla/platform-tilt/issues/3. 23

[JKL21] Hyeran Jeon, Nima Karimian, and Tamara Lehman. A New Foe in GPUs: Power Side-Channel Attacks on Neural Network. In *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, pages 313–313, April 2021. URL: https://ieeexplore.ieee.org/document/9424358, doi:10.1109/ISQED51717.2021.9424358. 20

[JM96] Heeralal Janwa and Oscar Moreno. McEliece Public Key Cryptosystems Using Algebraic-Geometric Codes. *Designs, Codes and Cryptography*, 8(3):293–307, June 1996. doi:10.1023/A:1027351723034. 33

[Jé23] Jérémy Breton. Leveraging Android Permissions: A Solver Approach. In *SSTIC 2023*, Rennes, 2023. URL: https://www.sstic.org/2023/presentation/leveraging_android_permissions/. 26

[KFG+19] Zijo Kenjar, Tommaso Frassetto, David Gens, Michael Franz, and Ahmad-Reza Sadeghi. V0LTpwn: Attacking x86 Processor Integrity from Software, December 2019. URL: http://arxiv.org/abs/1912.04870, arXiv:1912.04870, doi:10.48550/arXiv.1912.04870. 8

[KJFK18] Elmira Karimi, Zhen Hang Jiang, Yunsi Fei, and David Kaeli. A Timing Side-Channel Attack on a Mobile GPU. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pages 67–74, October 2018. URL: https://ieeexplore.ieee.org/abstract/document/8615670, doi:10.1109/ICCD.2018.00020. 20

[KJJR11] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, April 2011. doi:10.1007/s13389-011-0006-y. 21

[KM22] Fatima Khalid and Ammar Masood. Vulnerability analysis of Qualcomm Secure Execution Environment (QSEE). *Computers & Security*, 116:102628, May 2022. URL: https://www.sciencedirect.com/science/article/pii/S016740482200027X, doi:10.1016/j.cose.2022.102628. 9

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer. doi:10.1007/3-540-68697-5_9. 66

[KW16] Uri Kanonov and Avishai Wool. Secure Containers in Android: The Samsung KNOX Case Study, May 2016. URL: http://arxiv.org/abs/1605.08567, arXiv:1605.08567, doi:10.48550/arXiv.1605.08567. 24

[LDL+21] Rui Li, Wenrui Diao, Zhou Li, Jianqi Du, and Shanqing Guo. Android Custom Permissions Demystified: From Privilege Escalation to Design Shortcomings. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 70–86, May 2021. URL: https://ieeexplore.ieee.org/document/9519385, doi:10.1109/SP40001.2021.00070. 26

[Led] Ledger-Donjon. Rainbow. Ledger Donjon. URL: https://github.com/Ledger-Donjon/rainbow. 20

[LKO+21] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 355–371, May 2021. URL: https://ieeexplore.ieee.org/document/9519416, doi:10.1109/SP40001.2021.00063. 8

[LLD21] Ximing Liu, Yingjiu Li, and Robert H. Deng. UltraPIN: Inferring PIN Entries via Ultrasound. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ASIA CCS '21, pages 944–957, New York, NY, USA, May 2021. Association for Computing Machinery. doi:10.1145/3433210.3453075. 26

[low] lowRISC CIC. OpenTitan home page. URL: https://opentitan.org/. 84

[LWP20] Wenzheng Liu, Xiaofeng Wang, and Wei Peng. State of the Art: Secure Mobile Payment. *IEEE Access*, 8:13898–13914, 2020. URL: https://ieeexplore.ieee.org/document/8947955, doi:10.1109/ACCESS.2019.2963480. 11

[MB09] Rafael Misoczki and Paulo S. L. M. Barreto. Compact McEliece Keys from Goppa Codes. In Michael J. Jacobson, Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography*, Lecture Notes in Computer Science, pages 376–392, Berlin, Heidelberg, 2009. Springer. doi:10.1007/978-3-642-05445-7_24. 33

[MBH+20] Daniel Moghimi, Jo Van Bulck, Nadia Heninger, Frank Piessens, and Berk Sunar. {CopyCat}: Controlled {Instruction-Level} Attacks on Enclaves. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 469–486, 2020. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi-copycat. 8

[McE78] R.J. McEliece. A Public-Key Cryptosystem Based On Algbraic Coding Theory. 1978. xv, 30, 49

[MGH09] Wil Michiels, Paul Gorissen, and Henk D. L. Hollmann. Cryptanalysis of a Generic Class of White-Box Implementations. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography*, Lecture Notes in Computer Science, pages 414–428, Berlin, Heidelberg, 2009. Springer. doi:10.1007/978-3-642-04159-4_27. 20

[Mic] Microsoft Authenticator home page. URL: https://www.microsoft.com/en-us/security/mobile-authenticator-app. 6

[MM24] David McCabe and Tripp Mickle. U.S. Sues Apple, Accusing It of Maintaining an iPhone Monopoly. *The New York Times*, March 2024. URL: https://www.nytimes.com/2024/03/21/technology/apple-doj-lawsuit-antitrust.html. 84

[MOG+20] 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 *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1466–1482, May 2020. URL: https://ieeexplore.ieee.org/document/9152636, doi:10.1109/SP40000.2020.00057. 8

[MRPM11] David M'Raihi, Johan Rydell, Mingliang Pei, and Salah Machani. TOTP: Time-Based One-Time Password Algorithm. Request for Comments RFC 6238, Internet Engineering Task Force, May 2011. URL: https://datatracker.ietf.org/doc/rfc6238, doi:10.17487/RFC6238. 6

[MS07] Lorenz Minder and Amin Shokrollahi. Cryptanalysis of the Sidelnikov Cryptosystem. In Moni Naor, editor, *Advances in Cryptology - EUROCRYPT 2007*, Lecture Notes in Computer Science, pages 347–360, Berlin, Heidelberg, 2007. Springer. doi:10.1007/978-3-540-72540-4_20. 33

[MSSS11] H. Gregor Molter, Marc Stöttinger, Abdulhadi Shoufan, and Falko Strenzke. A simple power analysis attack on a McEliece cryptoprocessor. *Journal of Cryptographic Engineering*, 1(1):29–36, April 2011. doi:10.1007/s13389-011-0001-3. 33

[MSW16] Tarjei Mandt, Mathew Solnik, and David Wang. Demystifying the Secure Enclave Processor. *Black Hat 2016*, 2016. 10

[Mui13] James A. Muir. A Tutorial on White-box AES, 2013. URL: https://eprint.iacr.org/2013/104. 17

[Nal22] Petr Nalevka. Don't Kill My App: Android survival guide, November 2022. URL: https://www.droidcon.com/2022/11/15/dont-kill-my-app-android-survival-guide/. xvii, 57

[Nat99] National Institute of Standards and Technology (NIST). Data Encryption Standard (DES). Federal Information Processing Standard (FIPS) 46-3 (Withdrawn). Technical Report Federal Information Processing Standard (FIPS) 46-3 (Withdrawn), U.S. Department of Commerce, October 1999. URL: https://csrc.nist.gov/pubs/fips/46-3/final. 7

[Nat17] National Institute of Standards and Technology (NIST). Post-Quantum Cryptography Standardization, January 2017. URL: https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization. 32

[Nat23] National Institute of Standards and Technology (NIST). Module-Lattice-Based Key-Encapsulation Mechanism Standard. Technical Report Federal Information Processing Standard (FIPS) 203 (Draft), U.S. Department of Commerce, August 2023. URL: https://csrc.nist.gov/pubs/fips/203/ipd, doi:10.6028/NIST.FIPS.203.ipd. 51

[New] New Intel chips won't play Blu-ray disks due to SGX deprecation. URL: https://www.bleepingcomputer.com/news/security/new-intel-chips-wont-play-blu-ray-disks-due-to-sgx-deprecation/. 8

[NFK21] Shintaro Narisada, Kazuhide Fukushima, and Shinsaku Kiyomoto. Fast GPU Implementation of Dumer's Algorithm Solving the Syndrome Decoding Problem. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 971–977, New York City, NY, USA, September 2021. IEEE. URL: https://ieeexplore.ieee.org/document/9644911/, doi:10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00136. 31

[NP03] Eric Brier Naccache, David and Pascal Paillier. Chemical Combinatorial Attacks on Keyboards, 2003. URL: https://eprint.iacr.org/2003/217. 26

[NS24] Stephen Nellis and Akash Sriram. Apple retreats in Epic feud, allows Fortnite return in EU. *Reuters*, March 2024. URL: https://www.reuters.com/technology/apple-reinstates-epic-games-developer-account-2024-03-08/. 84

[Ope] Open Mobile Alliance Ltd. WAP home page. URL: https://www.wapforum.org/what/. 4

[Ope09] Open Mobile Terminal Platform. ADVANCED TRUSTED ENVIRONMENT: OMTP TR1, 2009. URL: http://www.omtp.org/OMTP_Advanced_Trusted_Environment_OMTP_TR1_v1_1.pdf. 8

[OTD10] Ayoub Otmani, Jean-Pierre Tillich, and Léonard Dallot. Cryptanalysis of Two McEliece Cryptosystems Based on Quasi-Cyclic Codes. *Mathematics in Computer Science*, 3(2):129–140, April 2010. doi:10.1007/s11786-009-0015-8. 33

[PGD+22] Sabine Pircher, Johannes Geier, Julian Danner, Daniel Mueller-Gritschneder, and Antonia Wachter-Zeh. Key-Recovery Fault Injection Attack on the Classic McEliece KEM, 2022. URL: https://eprint.iacr.org/2022/1529. 33

[Pie84] Pierre Bourdieu. *Homo Academicus*. Le sens commun. Minuit edition, 1984. vi

[PLHQ20] Sourav Panda, Yuanzhen Liu, Gerhard Petrus Hancke, and Umair Mujtaba Qureshi. Behavioral Acoustic Emanations: Attack and Verification of PIN Entry Using Keypress Sounds. page 26, 2020. 26, 71

[PRD+15] Martin Petrvalsky, Tania Richmond, Milos Drutarovsky, Pierre-Louis Cayrel, and Viktor Fischer. Countermeasure against the SPA attack on an embedded McEliece cryptosystem. In *Microwave and Radio Electronics Week 2015*, page pp. 462, April 2015. URL: https://hal-ujm.archives-ouvertes.fr/ujm-01186632, doi:10.1109/RADIOELEK.2015.7129055. 35

[PZZQ16] Ying Peng, Mingxin Zhang, Jinlong Zheng, and Zhenjiang Qian. Research on Android Access Control Based on Isolation Mechanism. In *2016 13th Web Information Systems and Applications Conference (WISA)*, pages 231–235, September 2016. URL: https://ieeexplore.ieee.org/abstract/document/7878263, doi:10.1109/WISA.2016.53. 24

[Qua] Quarkslab. QuarkslaB Dynamic binary Instrumentation (QBDI) home page. URL: https://qbdi.quarkslab.com/. 20

[RBSC17] Emma Raszmann, Kyri Baker, Ying Shi, and Dane Christensen. Modeling stationary lithium-ion batteries for optimization and predictive control. In *2017 IEEE Power and Energy Conference at Illinois (PECI)*, pages 1–7, February 2017. URL: https://ieeexplore.ieee.org/document/7935755, doi:10.1109/PECI.2017.7935755. 58

[RD23] Renaud Lambert and Dominique Plihon. Est-ce vraiment la fin du dollar ? *Le Monde diplomatique*, pages 8–9, November 2023. URL: https://www.monde-diplomatique.fr/2023/11/LAMBERT/66270. 13

[RMR+21] Hany Ragab, Alyssa Milburn, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. CrossTalk: Speculative Data Leaks Across Cores Are Real. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1852–1867, May 2021. URL: https://ieeexplore.ieee.org/document/9519489, doi:10.1109/SP40001.2021.00020. 8

[Rom22] Romain Thomas. DroidGuard: A Deep Dive into SafetyNet. In *SSTIC 2022*, Rennes, 2022. URL: https://www.sstic.org/2022/presentation/droidguard_a_deep_dive_into_safetynet/. 20

[RRH15] Alejandro Pérez Ruiz, Mario Aldea Rivas, and Michael González Harbour. CPU Isolation on the Android OS for running Real-Time Applications. In *Proceedings of the 13th International Workshop on Java Technologies for Real-time and Embedded Systems*, JTRES '15, pages 1–7, New York, NY, USA, October 2015. Association for Computing Machinery. doi:10.1145/2822304.2822317. 24

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978. URL: https://dl.acm.org/doi/10.1145/359340.359342, doi:10.1145/359340.359342. xv, 30

[SAL22] Stéphane Legleye, Amandine Nougaret, and Louise Viard-Guillot. Insee Focus N° 259. 2022. URL: https://www.insee.fr/fr/statistiques/6036909. xi, 79

[Sam] Samsung Pay home page. URL: https://www.samsung.com/us/samsung-pay/. 11

[Sam16] Samsung. Device-side Security: Samsung Pay, TrustZone, and the TEE, 2016. URL: https://developer.samsung.com/tech-insights/pay/device-side-security. 11

[SAS+19] Stephan van Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. RIDL: Rogue In-Flight Data Load. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 88–105, May 2019. URL: https://ieeexplore.ieee.org/document/8835281, doi:10.1109/SP.2019.00087. 8

[SCHK18] Stephan van Schaik, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Malicious Management Unit: Why Stopping Cache Attacks in Software is Harder Than You Think. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 937–954, 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/van-schaik. 22

[SeL] seL4 home page. URL: https://sel4.systems/. 84

[Sho94] P.W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, November 1994. doi:10.1109/SFCS.1994.365700. 33

[Sid94] V. M. Sidelnikov. A public-key cryptosystem based on binary Reed-Muller codes. 4(3):191–208, January 1994. URL: https://www.degruyter.com/document/doi/10.1515/dma.1994.4.3.191/html, doi:10.1515/dma.1994.4.3.191. 33

[SLM+19] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. ZombieLoad: Cross-Privilege-Boundary Data Sampling. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, pages 753–768, New York, NY, USA, November 2019. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/3319535.3354252, doi:10.1145/3319535.3354252. 8

[SMA+20] Stephan van Schaik, Marina Minkin, Andrew Kwong, Daniel Genkin, and Yuval Yarom. CacheOut: Leaking Data on Intel CPUs via Cache Evictions, June 2020. URL: http://arxiv.org/abs/2006.13353, arXiv:2006.13353, doi:10.48550/arXiv.2006.13353. 8

[SS92] V. M. Sidelnikov and S. O. Shestakov. On insecurity of cryptosystems based on generalized Reed-Solomon codes. 2(4):439–444, January 1992. URL: https://www.degruyter.com/document/doi/10.1515/dma.1992.2.4.439/html, doi:10.1515/dma.1992.2.4.439. 33

[SSMS10] Abdulhadi Shoufan, Falko Strenzke, H. Gregor Molter, and Marc Stöttinger. A Timing Attack against Patterson Algorithm in the McEliece PKC. In Donghoon Lee and Seokhie Hong, editors, *Information, Security and Cryptology – ICISC 2009*, Lecture Notes in Computer Science, pages 161–175, Berlin, Heidelberg, 2010. Springer. doi:10.1007/978-3-642-14423-3_12. 33

[STM+08] Falko Strenzke, Erik Tews, H. Gregor Molter, Raphael Overbeck, and Abdulhadi Shoufan. Side Channels in the McEliece PKC. In Johannes Buchmann and Jintai Ding, editors, *Post-Quantum Cryptography*, Lecture Notes in Computer Science, pages 216–229, Berlin, Heidelberg, 2008. Springer. doi:10.1007/978-3-540-88403-3_15. 33

[Str10] Falko Strenzke. A Timing Attack against the Secret Permutation in the McEliece PKC. In Nicolas Sendrier, editor, *Post-Quantum Cryptography*, Lecture Notes in Computer Science, pages 95–107, Berlin, Heidelberg, 2010. Springer. doi:10.1007/978-3-642-12929-2_8. 33

[Str11a] Falko Strenzke. Message-aimed side channel and fault attacks against public key cryptosystems with homomorphic properties. *Journal of Cryptographic Engineering*, 1(4):283–292, December 2011. doi:10.1007/s13389-011-0020-0. 33

[Str11b] Falko Strenzke. Timing Attacks against the Syndrome Inversion in Code-based Cryptosystems, 2011. URL: https://eprint.iacr.org/2011/683. 33

[SVSPF20] Jose Rodrigo Sanchez Vicarte, Benjamin Schreiber, Riccardo Paccagnella, and Christopher W. Fletcher. Game of Threads: Enabling Asynchronous Poisoning Attacks. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, pages 35–52, New York, NY, USA, March 2020. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/3373376.3378462, doi:10.1145/3373376.3378462. 8

[SWG+19] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Malware Guard Extension: Using SGX to Conceal Cache Attacks, May 2019. URL: http://arxiv.org/abs/1702.08719, arXiv:1702.08719, doi:10.48550/arXiv.1702.08719. 8

[SYG+19] Dimitrios Skarlatos, Mengjia Yan, Bhargava Gopireddy, Read Sprabery, Josep Torrellas, and Christopher W. Fletcher. MicroScope: Enabling microarchitectural replay attacks. In *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, pages 318–331, New York, NY, USA, June 2019. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/3307650.3322228, doi:10.1145/3307650.3322228. 8

[Tar23] Taro Yahagi. Anti-cheat and Anti-Piracy Measures in PC Games, Recommendations for In-House Production | CAPCOM Open Conference Professional RE:2023 | CAPCOM, 2023. URL: https://www.capcom-games.com/coc/2023/en/session/04/. 7

[Tez22] Cihangir Tezcan. Key lengths revisited: GPU-based brute force cryptanalysis of DES, 3DES, and PRESENT. *Journal of Systems Architecture*, 124:102402, March 2022. URL: https://www.sciencedirect.com/science/article/pii/S1383762122000066, doi:10.1016/j.sysarc.2022.102402. 16

[TF16] Bora Tar and Ayman Fayed. An overview of the fundamentals of battery chargers. In *2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1–4, October 2016. doi:10.1109/MWSCAS.2016.7870048. 58

[Tin] TinyOS home page. URL: http://tinyos.net/. 4

[TMM+13] Tancrède Lepoint, Matthieu Rivain, Mulder de Yoni, Bart Preneel, and Peter Roelse. Two Attacks on a White-Box AES Implementation. In *SAC 2013 - 20th International Conference Selected Areas in Cryptography*, volume 8282, page 265. Springer, August 2013. URL: https://inria.hal.science/hal-00872844, doi:10.1007/978-3-662-43414-7_14. 20

[Tru23] Trustonic. Kinibi-600a Commercial Release, August 2023. URL: https://www.trustonic.com/news/kinibi-600a-commercial-release/. 9

[Val70] Valerii Denisovich. A New Class of Linear Correcting Codes. *Problemy Peredachi Informatsii*, 6(3):24–30, 1970. xv, 30

[VMG14] Ingo Von Maurich and Tim Güneysu. Towards Side-Channel Resistant Implementations of QC-MDPC McEliece Encryption on Constrained Devices. In Michele Mosca, editor, *Post-Quantum Cryptography*, volume 8772, pages 266–282. Springer International Publishing, Cham, 2014. URL: http://link.springer.com/10.1007/978-3-319-11659-4_16, doi:10.1007/978-3-319-11659-4_16. 33

[VxW] VxWorks home page. URL: https://www.windriver.com/products/vxworks. 4

[WKPK16] Nico Weichbrodt, Anil Kurmus, Peter Pietzuch, and Rüdiger Kapitza. AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves. In Ioannis Askoxylakis, Sotiris Ioannidis, Sokratis Katsikas, and Catherine Meadows, editors, *Computer Security – ESORICS 2016*, Lecture Notes in Computer Science, pages 440–457, Cham, 2016. Springer International Publishing. doi:10.1007/978-3-319-45744-4_22. 8

[WL21] Leo Wildfeuer and Markus Lienkamp. Quantifiability of inherent cell-to-cell variations of commercial lithium-ion batteries. *eTransportation*, 9:100129, August 2021. URL: https://www.sciencedirect.com/science/article/pii/S2590116821000278, doi:10.1016/j.etran.2021.100129. 59

[WMGP07] Brecht Wyseur, Wil Michiels, Paul Gorissen, and Bart Preneel. Cryptanalysis of White-Box DES Implementations with Arbitrary External Encodings. In Carlisle Adams, Ali Miri, and Michael Wiener, editors, *Selected Areas in Cryptography*, Lecture Notes in Computer Science, pages 264–277, Berlin, Heidelberg, 2007. Springer. doi:10.1007/978-3-540-77360-3_17. 20

[Wor] World Wide Web Consortium. Web Authentication: An API for accessing Public Key Credentials. URL: www.w3.org/TR/webauthn/. 6

[XBZ12] Zhi Xu, Kun Bai, and Sencun Zhu. TapLogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors. In *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WISEC '12, pages 113–124, New York, NY, USA, April 2012. Association for Computing Machinery. doi:10.1145/2185448.2185465. 26

[XLL+15] Lei Xu, Guoxi Li, Chuan Li, Weijie Sun, Wenzhi Chen, and Zonghui Wang. Condroid: A Container-Based Virtualization Solution Adapted for Android Devices. In *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 81–88, March 2015. URL: https://ieeexplore.ieee.org/document/7130872, doi:10.1109/MobileCloud.2015.9. 24

[YGH17] Yuval Yarom, Daniel Genkin, and Nadia Heninger. CacheBleed: A timing attack on OpenSSL constant-time RSA. *Journal of Cryptographic Engineering*, 7(2):99–112, June 2017. doi:10.1007/s13389-017-0152-y. 20

[ZBJV20] Jie Zhou, Jian Bai, Meng Shan Jiang, and Fulvio Valenza. White-Box Implementation of ECDSA Based on the Cloud Plus Side Mode. *Security and Communication Networks*, 2020, January 2020. doi:10.1155/2020/8881116. 22

[ZHH+18] Yudi Zhang, Debiao He, Xinyi Huang, Ding Wang, and Kim-Kwang Raymond Choo. White-Box Implementation of the Identity-Based Signature Scheme in the IEEE P1363 Standard for Public Key Cryptography. Technical Report 814, 2018. URL: https://eprint.iacr.org/2018/814. 21

[ZK19] Wen-hai Zhou and Fan-tong Kong. Electromagnetic Side Channel Attack against Embedded Encryption Chips. In *2019 IEEE 19th International Conference on Communication Technology (ICCT)*, pages 140–144, October 2019. URL: https://ieeexplore.ieee.org/document/8947185, doi:10.1109/ICCT46805.2019.8947185. 20

Dépôt légal : septembre 2024

RÉSUMÉ

Les fournisseurs de services ont de plus en plus numérisé les biens qu'ils mettent à disposition des utilisateurs, que ce soit dans le domaine de la monétique, de l'identification ou des transports, entre autres. La conséquence directe est une forte concentration des contenus sensibles sur les appareils personnels, qu'il s'agisse de données ou d'exécutables, à la fois pour la vie privée et professionnelle.
Ces systèmes dits sur étagère sont en grande partie sous le contrôle des utilisateurs finaux. Les acteurs industriels et étatiques à la recherche d'environnements stables et de confiance rencontrent de nombreux défis lorsqu'ils cherchent à y déployer des solutions, puisqu'ils ne peuvent pas nécessairement exploiter les mécanismes de sécurité matériels, s'il y en a. Certains s'orientent alors vers des implémentations visant le modèle de sécurité en boîte blanche, impliquant que toutes les opérations puissent se dérouler à la vue de tous.
Les limites générales autour de ce paradigme sont abordées dans cette thèse. D'abord, la robustesse du cryptosystème de McEliece dans un tel contexte, avec son algorithme asymétrique aujourd'hui pertinent pour différentes raisons, a été étudiée, menant à la découverte d'une attaque sur sa clé privée basée sur l'injection de faute. Ensuite, la confidentialité des processus sur les plateformes Android a été remise en question. En particulier, il est montré comment, en instrumentalisant la gestion de l'énergie sur ces systèmes, une application malveillante peut en espionner une autre. Un cas pratique visant la récupération de codes numériques secrets tapés dans d'autres contextes est explicité. Des ajustements et améliorations ont malgré tout été proposés pour ces deux sujets.
Les travaux présentés contestent la vision hégémonique autour de la sécurité dans les produits informatiques sur étagère. Ceux-ci sont sous la responsabilité des utilisateurs mais peuvent également contenir des enclaves privatives régies par le secret et faisant office de racines de confiance. La robustesse des logiciels sensibles dépend alors de paramètres inéquitables tels que l'identité des développeurs, les ressources de l'utilisateur, et les choix techniques et personnels qu'il a réalisés. Une nouvelle organisation des responsabilités liées à la sécurité est proposée.

MOTS CLÉS

Sécurité ⋆ Système ⋆ Confiance ⋆ Boîte blanche ⋆ McEliece ⋆ Android

ABSTRACT

Services providers from a wide variety of domains (whether it is payments, identification, transport, or others) more and more digitalized the assets they provide to end users. In practice, this results in a strong concentration of sensitive contents (both data and executables) on people's personal smartphones, for their private and work life simultaneously.
This type of systems, often called Commercial Off-The-Shelf (COTS) devices, are under the control of end users. Industrial and state actors seeking trustable, stable execution environments thus face many challenges when trying to deploy solutions on them, as they do not necessarily have the possibility to exploit the hardware security mechanisms, if any. In consequence, they sometimes try to develop software operating in the white-box security model, where everything has to operate under open scrutiny.
In this thesis, the general limits of such a paradigm are addressed. First, it is showed how the McEliece cryptosystem, and its asymmetrical algorithm with varied advantages today, can be vulnerable in such a context by demonstrating an attack based on fault injection against it. Secondly, a focus is provided on the confidentiality guarantees provided by Android platforms. Specifically, by inspecting the implementation of power management in these, it is demonstrated how a malicious application can spy on a legitimate one. A concrete given example is focused on the recovery of Personal Identification Numbers codes entered in other software contexts. In both of these subjects, possible improvements and adjustments are nevertheless described.
These works intend to question the way security is envisioned on people's COTS devices. These are under the responsibility of end users, but can contain private parts based on secrets as roots of trust. The robustness of sensitive software thus depends on unfair parameters such as who is the developer, what can afford the end users, and what technical and personal choices has he made. A new distribution of security responsibilities is finally proposed.

KEYWORDS

Security ⋆ System ⋆ Trust ⋆ White-box ⋆ McEliece ⋆ Android