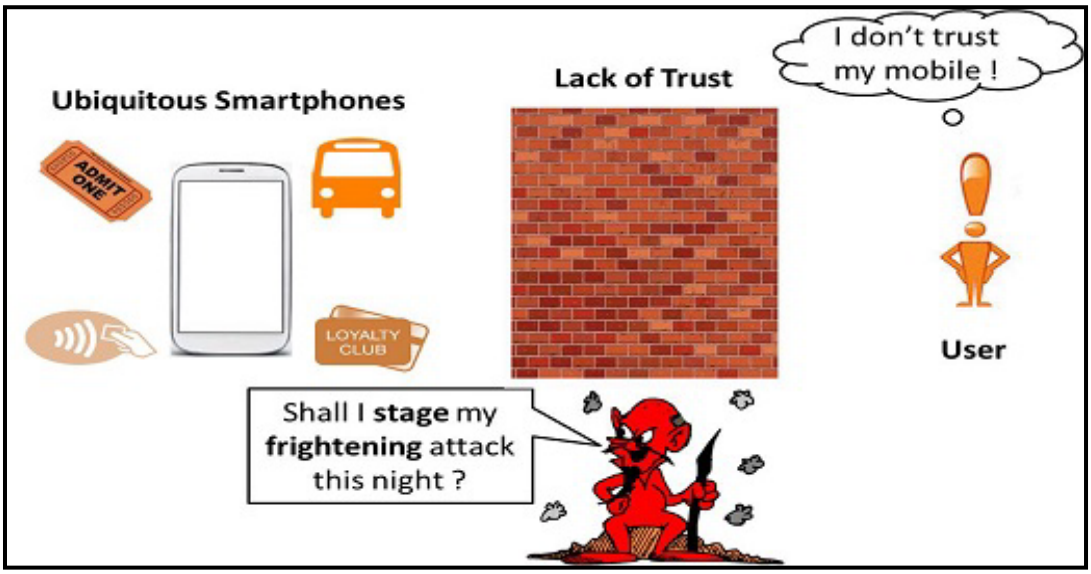
Par **Mohamed SABT**

Outsmarting smartphones : trust based on provable security and hardware primitives in smartphones architectures

Thèse présentée
pour l'obtention du grade
de Docteur de l'UTC



Soutenue le 13 décembre 2016
Spécialité : Informatique et Sciences et Technologies de l'Information et des Systèmes : Unité de recherche Heudyasic (UMR-7253)

D2320

THÈSE

pour obtenir le grade de

Docteur de l'Université de Technologie de Compiègne

Spécialité : Informatique et Sciences et Technologies de l'Information et des Systèmes

Présentée et soutenue par

Mohamed SABT

le 13 décembre 2016

Outsmarting Smartphones : Trust Based on Provable Security and Hardware Primitives in Smartphones Architectures

---

Devant le jury composé de :

Directeur de thèse :	Abdelmadjid Bouabdallah	(UTC)
Encadrants :	Mohammed Achemlal	(ENSICAEN)
	Jacques Traoré	(Orange Labs)
Rapporteurs :	Pascal Urien	(Telecom ParisTech)
	Henri Gilbert	(ANSSI)
Examinateurs :	Isabelle Chrisment	(Loria)
	Aurélien Francillon	(EURECOM)
	Dritan Nace	(UTC)

Travaux effectués au sein de l'Équipe NPS des Orange Labs
et de l'Équipe RO du laboratoire Heudiasyc à l'UTC

Abstract

The landscape of mobile devices has been changed with the introduction of smartphones. Since their advent, smartphones have become almost vital in the modern world. This has spurred many service providers to propose access to their services via mobile applications. Despite such big success, the use of smartphones for sensitive applications has not become widely popular. The reason behind this is that users, being increasingly aware about security, do not trust their smartphones to protect sensitive applications from attackers. The goal of this thesis is to strengthen users trust in their devices. We cover this *trust problem* with two complementary approaches: provable security and hardware primitives.

In the first part, our goal is to demonstrate the limits of the existing technologies in smartphones architectures. To this end, we analyze two widely deployed systems in which careful design was applied in order to enforce their security guarantee: the Android KeyStore, which is the component shielding users cryptographic keys in Android smartphones, and the family of Secure Channel Protocols (SCPs) defined by the GlobalPlatform consortium. Our study relies on the paradigm of provable security. Despite being perceived as rather theoretical and abstract, we show that this tool can be handily used for real-world systems to find security vulnerabilities. This shows the important role that can play provable security for *trust* by being able to formally prove the absence of security flaws or to identify them if they exist.

The second part focuses on complex systems that cannot cost-effectively be formally verified. We begin by investigating the dual-execution-environment approach. Then, we consider the case when this approach is built upon some particular hardware primitives, namely the ARM TrustZone, to construct the so-called Trusted Execution Environment (TEE). Finally, we explore two solutions addressing some of the TEE limitations. First, we propose a new TEE architecture that protects its sensitive data even when the secure kernel gets compromised. This relieves service providers of fully trusting the TEE issuer. Second, we provide a solution in which TEE is used not only for execution protection, but also to guarantee more elaborated security properties (i.e. self-protection and self-healing) to a complex software system like an OS kernel.

Résumé

Le paysage du monde des téléphones mobiles a changé avec l'introduction des ordiphones (de l'anglais smartphones). En effet, depuis leur avènement, les ordiphones sont devenus incontournables dans des différents aspects de la vie quotidienne. Cela a poussé de nombreux fournisseurs de services de rendre leurs services disponibles sur mobiles. Malgré cette croissante popularité, l'adoption des ordiphones pour des applications sensibles n'a toujours pas eu un grand succès. La raison derrière cela est que beaucoup d'utilisateurs, de plus en plus concernés par la sécurité de leurs appareils, ne font pas confiance à leur ordiphone pour manipuler leurs données sensibles. Cette thèse a pour objectif de renforcer la confiance des utilisateurs en leur mobile. Nous abordons ce *problème de confiance* en suivant deux approches complémentaires, à savoir la sécurité prouvée et la sécurité ancrée à des dispositifs matériels.

Dans la première partie, notre objectif est de montrer les limitations des technologies actuellement utilisées dans les architectures des ordiphones. À cette fin, nous étudions deux systèmes largement déployés et dont la sécurité a reçu une attention particulière dès la conception : l'entrepôt de clés d'Android, qui est le composant protégeant les clés cryptographiques stockées sur les mobiles d'Android, et la famille des protocoles sécurisés SCP (de l'anglais Secure Channel Protocol) qui est définie par le consortium GlobalPlatform. Nos analyses se basent sur le paradigme de la sécurité prouvée. Bien qu'elle soit perçue comme un outil théorique voire abstrait, nous montrons que cet outil pourrait être utilisé afin de trouver des vulnérabilités dans des systèmes industriels. Cela atteste le rôle important que joue la sécurité prouvée pour la *confiance* en étant capable de formellement démontrer l'absence de failles de sécurité ou éventuellement de les identifier quand elles existent.

Quant à la deuxième partie, elle est consacrée aux systèmes complexes qui ne peuvent pas être formellement vérifiés de manière efficace en termes de coût. Nous commençons par examiner l'approche à double environnement d'exécution. Ensuite, nous considérons le cas où cette approche est instanciée par des dispositifs matériels particuliers, à savoir le ARM TrustZone, afin de construire un *environnement d'exécution de confiance* (TEE de l'anglais Trusted Execution Environment). Enfin, nous explorons deux solutions palliant quelques limitations actuelles du TEE. Premièrement, nous concevons une nouvelle architecture du TEE qui en protège les données sensibles même quand son noyau sécurisé est compromis. Cela soulage les fournisseurs des services de la contrainte qui consiste à faire pleinement confiance aux fournisseurs du TEE. Deuxièmement, nous proposons une solution dans laquelle le TEE n'est pas uniquement utilisé pour protéger l'exécution des applications sensibles, mais aussi pour garantir à des grands composants logiciels (comme le noyau d'un système d'exploitation) des propriétés de sécurité plus complexes, à savoir l'auto-protection et l'auto-remédiation.

تَوْطِئَة

مازال عالم أجهزة الهواتف المحمولة يشهد الكثير من التحولات مع قدوم الهواتف الذكية؛ فمنذ مجيئها أصبحت هذه الهواتف جزءاً أساسياً من حياتنا اليومية مما دفع العديد من مزودي الخدمات لتوفير تطبيقات هاتفية من أجل جلب أكبر عدد من الزبائن. ولكن، وعلى الرغم من هذا النجاح الباهر، فقد ظل استخدام الهواتف الذكية في مجال التطبيقات الحساسة (كتطبيقات الدفع عن طريق المحمول) محدود الانتشار، فمعظم مستخدمي هذه الهواتف يرفضون ائتمان أجهزتهم على معلوماتهم الحساسة بسبب ما يصل إلى مسامعهم عن ثغرات أمنية يتم اكتشافها بشكل دوري. الهدف من بحث الدكتوراه هذا هو تعزيز ثقة المستخدمين بهواتفهم الذكية وذلك باتباع طريقتين مختلفتين تتمم الأولى منهما الثانية، هاتان الطريقتان هما الحماية المُبرهَنة (**provable security**) وتقنية الأجهزة العتادية (**hardware primitives**) المخصصة للحماية.

تنقسم رسالة الدكتوراه إلى قسمين أساسيين، الهدف من القسم الأول هو إثبات محدودية التقنيات المستخدمة حالياً لحماية الهواتف الذكية؛ لذلك يتناول هذا القسم دراسة تحليلية لنظامَيْ حماية واسعَيْ الانتشار، النظام الأول هو مستودع المفاتيح (**Android KeyStore**) والذي يعمل كدرع واقٍ لمفاتيح التشفير السرية في نظام التشغيل أندرويد، أما الثاني فهي مجموعة بروتوكولات التشفير (**Secure Channel Protocols**) المستخدمة في عالم البطاقات الذكية. تستعير دراستنا التحليلية أطرها من مبادئ الحماية المبرهَنة والتي قمنا بالاعتماد عليها من أجل العثور على ثغرات أمنية، نجاحنا في هذا المجال يبين الدور الهام الذي يمكن أن تلعبه الحماية المبرهَنة لأنظمة الهواتف الذكية على الرغم من كونها تُنظر في كثير من الأحيان على أنها نظرية ومجردة.

أما القسم الثاني فهو مخصص لبحث الأنظمة المعقدة والتي لا يمكن التأكد من شمولية أو تمامية نظم حمايتها بطريقة فعالة، يبدأ هذا القسم بدراسة تفصيلية للمنهج ثنائي البيئة الحَوسبية (**Dual-execution-environment**) ، يليها تركيز معمق على الحالة الخاصة التي يتم فيها بناء هذا المنهج باستخدام أجهزة عتادية معينة تسمى **ARM TrustZone** ، وذلك لإنشاء ما يُطلق عليه بالبيئة الحَوسبية الموثوقة (**TEE: Trusted Execution Environment**). الهدف من هذا التركيز هو التعرف الجيد على هذه البيئة الحوسبية من أجل إعادة التفكير بتصاميمها الحالية؛ وذلك للتغلب على بعض القيود التي تحد من انتشار هذه التقنية. وفي هذا الإطار قمنا بإنشاء حلَّيْن جديدَيْن: الأول هو عبارة عن بيئة حوسبية موثوقة يمكنها حماية بياناتها الحساسة حتى وإن تم اختراق الجزء الأكبر من نظام حمايتها، أما الحل الثاني فهو عبارة عن تعزيز مجال استخدام هذه البيئة ليشمل ليس فقط توفير الحماية للتطبيقات الحساسة ولكن أيضاً تأمين خصائص حمائية أكثر دقة (مثل الدفاع والعلاج الذاتي) لأنظمة حوسبية معقدة كنواة أنظمة التشغيل في الأجهزة المدمجة.

Acknowledgments

Thesis is like desert. Many never suspect that it can, despite its peaceful emptiness and infinite silence, reveal countless surprises to its inhabitants[1]. Only the people who have triumphantly crossed this seemingly simple place know that no one could ever survive alone and without any guidance or company. In the following few lines, I would like to express my gratitude towards those whose help and support were invaluable for me in order to successfully complete this PhD thesis.

First and foremost, I would like to thank my PhD adviser, Abdelmadjid Bouabdallah, for letting me write my master's thesis under his supervision, and then encouraging me to continue in research as his PhD student. It has been great working with him during the past years. I would also like to thank my two advisers in Orange Labs: Mohammed Achemlal and Jacques Traoré. Thank you Mohammed for your pleasant assistance in my first steps in research. I appreciate our numerous discussions, your myriad comments when proofreading my papers, and your constant willingness to share your clear view on a wide range of subjects. Under your guidance, I have improved tremendously, both as a person and as a scientist. On the same note, I am deeply grateful to Jacques in which one can find a source of wisdom and an oracle of knowledge. Thanks for making me discover a world of magic; a world full of Games, Oracles and Alice (aka Cryptography). I shall admit that I have learnt a lot from your impressive scientific rigor as well as your refined taste for research.

I would like to thank those who have participated in the validation of my thesis work. I am quite honored that Pascal Urien and Henri Gilbert have spent some of their limited time proofreading this dissertation. Many thanks go to Aurélien Francillon, Isabelle Chrisment and Dritan Nace for accepting being part of my dissertation committee.

I conducted my PhD research work in Orange Labs in Caen. I am pleased that Orange Labs as a whole, with its dynamic composition and members, was a nice environment for discussion and collaboration with other experts. In particular, I would like to thank all the members of the NPS (Networks and Products Security) team with whom I had the privilege to work on some exciting research. Special thanks go to Jean-François Misarsky; the head of the NPS team. Thank you Jean-François for creating a great atmosphere that keeps the cohesion of the team. I am convinced that this[2] has been one of the main reasons behind the high quality research work produced by NPS.

I have many reasons to express my recognition to (almost) every single NPS member. Here, I only mention a very few – the others whom I do not mention will forgive me so that I remain brief. Let me start by three (ex-)members: Mouhannad Alattar, Ghada Arfaoui and Nabil Boujenane. I appreciated collaborating with Mouhannad in various research projects. I owe him great thanks for his big help in filing a patent application about the NFC technology. My thanks go to Ghada who was the first to explore TEE in literature and whose work was really handy to make a good start. As for Nabil, an ex-intern, I wish to thank him for his hard work on ARM TrustZone. Nabil has changed my perception by showing me the huge difference between drawing

[1] *Abdul Rahman Munif*, Cities of Salt, Al-Munbatt (The Uprooted)

[2] additionally to the fact that they work just next to 'Le Silicon Valley'

fancy TEE architectures and implementing them on real hardware. I sincerely thank my special proofreading group formed by some of the NPS PhD students: Amira, Julien, Marie, Quentin and Solenn. Most of my papers have been voluntarily revised by at least one of them, and their often-peculiar comments have indeed helped improve the clarity of my writing style.

I would like to thank all the members of the O'Sec team. I really enjoyed our experience in the 19th edition of the *Armor Cup*; the Regatta of the Orange Group. I am grateful to Julien Hatin, our cox, for his patience and for never giving up training us in order to make this experience successful. I also thank Olivier Aunay who took on his responsibility the good organization of the O'Sec team. It will remain vivid forever in my memory, that is the magnificent view of the colorful yacht fleet on the blue ocean waves in Brittany.

Even though, for practical reasons, I seldom was at the UTC, I was honored to be part of this institution. I would like to thank Ali Charara, the director of the UTC-Heudiasyc laboratory, for creating great conditions to do research. I wish to thank the Heudiasyc PhD students, although I did not get to meet them all, who made me live the UTC atmosphere by keeping including me in their email list. My warm thanks go to one of them, Amira, who performed her thesis in Orange Labs and who happened to end up in the same team as mine. Amira, there is much stuff that we both had to do together, and I am grateful that you were such a big help.

I would like to thank the "first generation" of PhD students that I knew in Orange Labs: Chrystel , Hachem and Kahina. Guys, you will never believe how helpful your rantings were for me to better face my thesis. I appreciated your friendship and your never-ending discussions about thesis, research, PhDcomics, free food, and even about the role of (central) governments to build strong states (well, the last one was actually mine). You can see how much I have improved, as I forced myself to limit the use of my favorite expression, namely "*it is worth noting/mentioning*", to only 14 times in my dissertation. You can imagine how hard it was for me not to include more.

I would also like to thank the "triple-M generation" of PhD students that lucky coincidence made us start our thesis at the same time and in the same place: Marie and Mayla. I believe that sharing this journey together, with its ups and downs, have made of us more than colleagues more like true friends. Thank you for being a part of my journey and thank you for allowing me to be a part of yours. Thank you for your time, helps, conversations and laughter. My thesis would not have been the same without our friendship that was a precious support for me. I hope all the best for you for your future life.

Finally, all my sincere gratitude and indebtedness go to my dear family, especially mom, dad, brothers and sisters. Thank you for your endless encouragements. They were the light that brightened up my darkest nights. Thank you for your unconditional belief in me. Let me admit one truth: your deep confidence and trust obliged me to fake it, and I faked it until I made it[3].

I am aware that these short lines are not enough to convey my acknowledgments to all those who have helped me throughout my PhD journey. However, as goes the belief of the desert inhabitants[4], it is in the few words that one can find profound meanings. When writing our acknowledgments, we fast realize that language is poor and short of words to express our true feelings. A mere *"thank you"* might seem not enough, but our life experience has shown us that these two words possess some magic when they are said with heartfelt gratitude. So, may you all accept my grateful thanks with sincere recognition and heartfelt gratitude.

[3] *Amy Cuddy*, TED Talks, Your body language shapes who you are

[4] خَيْرُ الكَلَامِ مَا قَلَّ وَدَلَّ

"Above every man of knowledge, there is one who knows better"

إلى من جَلَّدَتْ كتبي أيام المدرسة لأنها بذلك علَّمتني حُبَّ العِلم واحترام حامله...
وإلى من علَّمني أنَّ كل إسراف قبيح ماعدا الإسراف في طلب العلم لأنَّ قيمة الإنسان عِلْمُه...

إهداء إلى أبي وأمي مع التحية.

Contents

Introduction (version française) **1**
1 Contexte . . . 1
2 Problématique de recherche . . . 2
3 Nos contributions . . . 3
4 Structure du manuscrit . . . 6

1 Introduction **7**
1.1 Motivation . . . 7
1.2 Research Topic . . . 8
1.3 Contributions . . . 9
1.4 Outline . . . 11

I Preliminaries **13**

2 Trust on Smartphones **15**
2.1 Important Note . . . 15
2.2 Smartphone Security . . . 16
2.2.1 Security Model . . . 16
2.2.2 Security Challenges . . . 18
2.3 Security Internals of Android . . . 20
2.3.1 Android Architecture . . . 20
2.3.2 Security Enforcement . . . 21
2.3.3 The Binder . . . 22
2.3.4 The Android KeyStore . . . 22
2.4 Secure Hardware Technologies in Smartphones . . . 24
2.4.1 The Baseband . . . 24
2.4.2 The Secure Element . . . 25
2.4.3 ARM TrustZone . . . 27
2.5 Conclusion . . . 29

3 Cryptographic Primitives and Provable Security **31**
3.1 Cryptographic Primitives . . . 32
3.1.1 Notations . . . 32
3.1.2 Block Ciphers . . . 32
3.1.3 Tweakable Block Ciphers . . . 33
3.1.4 Encryption Schemes and Mode of Operations . . . 33
3.1.5 Nonce-Based Symmetric Encryption . . . 36
3.1.6 Encoding Schemes . . . 37
3.1.7 Message Authentication Codes . . . 37
3.1.8 Authenticated Encryption . . . 38

3.2 Provable Security . . . 39
3.2.1 Overview of Provable Security . . . 40
3.2.2 Functions and Permutations . . . 42
3.2.3 Threat Models for Symmetric Encryption . . . 43
3.2.4 Confidentiality of Symmetric Encryption . . . 44
3.2.5 Integrity of Symmetric Encryption . . . 45
3.2.6 Indistinguishability from Random Bits . . . 46
3.2.7 Strong Unforgeability of MAC . . . 47
3.2.8 Results for Generic Compositions . . . 47

II Formal Security Analysis of Two Real-World Systems 49

4 Breaking Into the Android KeyStore 51
4.1 Responsible Disclosure . . . 52
4.2 Introduction . . . 52
4.2.1 Overview of our Attack . . . 53
4.2.2 Context in Trusted Computing . . . 53
4.3 Background . . . 53
4.4 Related Work . . . 54
4.5 Hash-then-Encrypt Security Results . . . 56
4.5.1 Integrity Notions . . . 56
4.5.2 Hash-then-Encrypt . . . 56
4.5.3 Hash-then-Encrypt is IND-CPA Secure . . . 56
4.5.4 Hash-then-CBC-Encrypt . . . 57
4.5.5 Hash-then-CTR-Encrypt . . . 59
4.6 The Android KeyStore . . . 60
4.6.1 KeyStore Service . . . 61
4.6.2 The Keys Formats . . . 61
4.7 Attacking the Android KeyStore . . . 62
4.7.1 Threat Model . . . 62
4.7.2 The Forgery Attack . . . 62
4.7.3 The Undetected Malware . . . 63
4.8 Discussion and Recommendations . . . 66
4.9 Conclusion . . . 67

5 Cryptanalysis of GlobalPlatform Secure Channel Protocols 69
5.1 Introduction . . . 70
5.1.1 Overview of our Attack . . . 70
5.1.2 Context in Trusted Computing . . . 71
5.2 Background . . . 71
5.2.1 Secure Card Content Management . . . 72
5.2.2 Secure Channel Protocols . . . 73
5.3 Related Work . . . 73
5.4 Secure Channel Protocol ‘2’ . . . 75
5.4.1 Description . . . 75
5.4.2 Try-and-Guess Attack . . . 75
5.4.3 Plaintext Recovery Against Smart Cards . . . 77
5.4.4 Discussion about Theoretical Feasibility . . . 78
5.5 Secure Channel Protocol ‘3’ . . . 79
5.5.1 Description . . . 79

5.5.2 Security Models . . . 81
5.6 SCP03 Security Results . . . 85
5.6.1 Sf-nEtTw Security Analysis . . . 85
5.6.2 SCP03 Security Analysis . . . 86
5.6.3 Practical Considerations of SCP03 Security Analysis . . . 89
5.7 Security Proofs of Sf-nEtTw . . . 90
5.7.1 Proof of Proposition 5.1 . . . 90
5.7.2 Proof of Theorem 5.1 . . . 93
5.8 Discussion . . . 95

III Trust on Smart Complex Systems 97

6 Towards the Dual-Execution-Environment Approach 99
6.1 Introduction . . . 100
6.1.1 Previous Work . . . 100
6.1.2 Overview of our Comparative Framework . . . 102
6.2 Background . . . 102
6.2.1 Trusted Model . . . 102
6.2.2 MILS . . . 103
6.3 Related Work . . . 104
6.4 The Dual-EE Approach . . . 104
6.4.1 From Dual-EE to Multi-EE . . . 105
6.4.2 Classification of Dual-EE Solutions . . . 106
6.5 Comparison Methodology . . . 107
6.5.1 Functional Criteria . . . 107
6.5.2 Security Criteria . . . 108
6.5.3 Deployability Criteria . . . 108
6.6 Comparative Evaluation . . . 109
6.6.1 External Hardware Module: Secure Element . . . 109
6.6.2 Bare-Metal Hypervisor: KVM/ARM . . . 109
6.6.3 Special Processor Extensions: TrustZone . . . 110
6.7 Discussion . . . 111
6.8 Conclusions . . . 112

7 Trusted Execution Environment: What It Is, and What It Is Not 115
7.1 Introduction . . . 116
7.1.1 Inconsistent Definitions . . . 117
7.1.2 Previous Attempts . . . 118
7.2 Background . . . 118
7.3 Trusted Execution Environment . . . 119
7.3.1 Prerequisite: Separation Kernel . . . 119
7.3.2 Definition . . . 119
7.3.3 Discussion . . . 119
7.3.4 How Trust Can Be Measured . . . 120
7.3.5 Related Concepts . . . 121
7.4 Building Blocks . . . 121
7.5 Inter-Environment Communication . . . 124
7.5.1 GlobalPlatform TEE Client API . . . 124
7.5.2 Trusted Language Runtime (TLR) . . . 125
7.5.3 SafeG Real-Time Secure RPC . . . 125

7.6 Trusted I/O Path . . . 125
7.6.1 GlobalPlatform Trusted User Interface API . . . 125
7.6.2 VeriUI . . . 126
7.6.3 TrustUI . . . 126
7.7 Secure Storage . . . 126
7.7.1 GlobalPlatform Trusted Storage API . . . 126
7.7.2 Android KeyStore . . . 127
7.8 Formal Methods . . . 127
7.9 ARM TrustZone-Based TEE . . . 128
7.9.1 Industrial TEEs . . . 128
7.9.2 Academic TEEs . . . 129
7.9.3 TrustZone Emulation Frameworks . . . 130
7.9.4 Comparative Study . . . 130
7.10 TEE Applications . . . 130
7.10.1 Ticketing . . . 130
7.10.2 Mobile Payment . . . 132
7.10.3 Media Content Protection . . . 132
7.10.4 Authentication . . . 132
7.10.5 NFC . . . 132
7.10.6 Trusted Sensors . . . 133
7.10.7 Trusted Platform Module . . . 133
7.10.8 Self-Protection . . . 133
7.11 Attacks on TEE . . . 134
7.12 Conclusion . . . 134

8 Reconsidering the Power of Trusted Execution Environment 137
8.1 Trusted Execution Environment Based on Garbled Circuits . . . 138
8.1.1 Introduction . . . 138
8.1.2 Background . . . 139
8.1.3 Two-Party Secure Computation . . . 139
8.1.4 Motivation and Threat Model . . . 141
8.1.5 TrustedGarble Design . . . 142
8.1.6 Development in TrustedGarble . . . 144
8.1.7 Security Analysis . . . 145
8.1.8 Open Issues . . . 145
8.1.9 Related Work . . . 146
8.1.10 Conclusion . . . 147
8.2 Towards Integrating TEE Into Autonomic Systems . . . 148
8.2.1 Introduction . . . 148
8.2.2 Autonomic Computing . . . 148
8.2.3 Self-Protection . . . 149
8.2.4 Self-Healing . . . 149
8.2.5 Problem Statement . . . 149
8.2.6 Threat Model and Assumptions . . . 150
8.2.7 Architecture . . . 151
8.2.8 Open Issues . . . 153
8.2.9 Conclusion . . . 154

9 Conclusion and Perspectives 155

List of Publications **158**

Patents **160**

List of Tables **161**

List of Figures **162**

Acronyms **167**

Bibliography **168**

Introduction

(The English version is given below)

"Le Lapin Blanc mit ses lunettes.
Par où commencerai-je, s'il plaît à Votre Majesté ?
demanda-t-il.
Commencez par le commencement, dit gravement le Roi,
et continuez jusqu'à ce que vous arriviez à la fin ; là, vous
vous arrêterez."

– Lewis Carroll, Alice au pays des merveilles

Sommaire

1	**Contexte**	**1**
2	**Problématique de recherche**	**2**
3	**Nos contributions**	**3**
4	**Structure du manuscrit**	**6**

Dans ce chapitre, nous présentons brièvement les travaux menés dans le cadre de cette thèse. Nous motivons dans un premier temps la nécessité de renforcer la sécurité des téléphones mobiles puis décrivons les problématiques sous-jacentes en termes de sécurité. Nous détaillons par la suite nos contributions apportées dans ce domaine. Nous concluons ce chapitre par présenter le sommaire de ce manuscrit.

1 Contexte

C'est en 2007 que les premiers ordiphones (de l'anglais smartphones) ont été introduits sur le marché par Apple qui a été vite suivi par Google en 2008. Depuis lors, les ordiphones sont devenus indispensables pour de nombreuses personnes. Selon une étude récente [IDC16b], le marché mondial des ordiphones a produit $1,48$ milliard d'appareils en 2016 dont 82,5% utilisent le système d'exploitation mobile Android. Les ordiphones doivent ce grand succès à leur petite taille qui les rend maniables et transportables, à leur capacité de calcul qui est suffisante pour assurer le bon fonctionnement d'un grand nombre d'applications, ainsi qu'à leur connectivité omniprésente qui permet aux utilisateurs de rester en ligne tout au long de la journée. Ces trois caractéristiques font des ordiphones un outil formidable qui nous aide à réaliser nos tâches quotidiennes.

La popularité des ordiphones est indirectement due aussi aux fournisseurs des services qui proposent une version mobile de leurs services, augmentant ainsi considérablement le nombre d'applications disponibles pour ces appareils. Par exemple, plus de 2 millions d'applications mobiles étaient disponibles pour le système d'Android en février 2016 [Sta16]. Bien que quelques applications soient payantes, l'écrasante majorité de celles-ci sont gratuites, ce qui a permis d'augmenter encore leur popularité.

Outre les services de communication de base, les ordiphones permettent également à leurs utilisateurs d'échanger des messages avec leurs amis, d'envoyer des courriels confidentiels à leurs collègues de travail, de gérer leurs comptes en banque, voire même de régler leurs achats. Beaucoup de tâches réalisées par les ordiphones requièrent la manipulation de données sensibles, telles que des données confidentielles ou celles liées à l'identification des utilisateurs. Cela pose de nombreuses questions en termes de sécurité, puisque la nature portable et connectée des ordiphones rend ces appareils particulièrement vulnérables à un large spectre d'attaques.

En effet, le fait d'être connectés et ouverts simplifie considérablement la tâche pour les attaquants. Par exemple, depuis sa première apparition en 2008, environ 540 failles de sécurité ont été signalées contre le système d'Android dans la base de données CVE [CVE16a]. Étant de plus en plus conscients de ces problèmes de sécurité, les utilisateurs hésitent encore à s'en servir pour exécuter des applications sensibles. Ce manque de confiance entrave de grands fournisseurs de services pour les ordiphones, comme Orange, dans leurs efforts pour continuer d'innover en proposant des nouveaux services ayant besoin d'un certain niveau de sécurité [Bro16]. Ainsi, le domaine de la sécurité des ordiphones est devenu une priorité pour ces fournisseurs. Pallier ce problème leur assurerait d'avoir davantage de clients qui utilisent leurs services.

2 Problématique de recherche

Récemment, les ordiphones sont devenus une cible privilégiée pour les attaquants qui cherchent à gagner de l'argent en compromettant les mobiles. En conséquence, le nombre de logiciels malveillants est en constante croissance depuis plusieurs années. Bien qu'il existe des travaux de recherche variés, provenant à la fois des mondes académique et industriel, qui ont produit différents mécanismes de sécurité répondant à ce grand défi, ils sont pour la plupart inefficaces en pratique à cause de plusieurs problèmes conceptuels concernant la sécurité de leurs architectures logicielles (en particulier, leur système d'exploitation) ainsi que de leurs architectures matérielles. Afin d'atteindre les objectifs de sécurité à moyen et à long terme, l'industrie des ordiphones a besoin de solutions qui présentent un bon rapport coût/efficacité et qui bien intègrent les mesures de sécurité dès la conception avec un fondement architectural adéquat.

Ainsi, il y a une tendance d'intégrer des concepts de *l'informatique de confiance* dans la conception des ordiphones. L'informatique de confiance est une branche particulière de la sécurité informatique. D'après Gasser [Gas88], un logiciel de confiance est un logiciel en charge d'assurer la sécurité du système, et qui est a fortiori exempt de faille de sécurité. Depuis son introduction, l'informatique de confiance a suscité plusieurs débats au sein de la communauté de sécurité. En effet, le terme "informatique de confiance" est assez trompeur. Dans son sens premier, il signifie qu'il existe une entité qui fait confiance au logiciel en question. Il est à noter qu'une telle affirmation omet de préciser de quelle entité il s'agit et sur quelle base la confiance de cette entité a été établie. Un des problèmes principaux dans ce domaine est que "la confiance" est une notion subjective, et donc non-mesurable. En français, selon le Petit Robert, la confiance relève "de la foi, de l'assurance et de l'espérance ferme". La notion de la confiance dans le domaine de l'informatique est plus subtile. Cependant, elle peut être inspirée par notre vie quotidienne: une personne de confiance est une personne qui se comporte comme prévu. En informatique, nous pouvons assumer qu'elle suit la même logique.

Dans les architectures complexes, il est impossible de faire confiance à tous les composants du système. En effet, ces systèmes sont structurés de telle manière à minimiser le nombre de composants ayant besoin de se faire confiance. Alors, au sein d'un seul système, il est généralement utile de distinguer les composants de confiance des autres. Un *composant de confiance* est un composant qui ne contient pas de faille afin d'assurer la sécurité du système, alors que les composants ne remplissant pas cette caractéristique englobent tout le reste. Cette manière de concevoir est très répandue pour environnements où des données sensibles sont manipulées. Dans

un monde idéal de sécurité, les composants de confiance seraient construits en deux étapes. La première étape serait réservée au développement assuré par des entités de confiance suivant un cahier des charges précis. Lors de la seconde étape, elle serait consacrée à la vérification des composants développés, via des outils issus des méthodes formelles. Cette manière de construire des composants dépasse de loin les procédés pratiqués actuellement dans l'industrie, du fait de son coût et de sa complexité de mise en œuvre.

Certains ont tendance à simplifier leur modèle de confiance en réduisant le composant de confiance aux dispositifs matériels de sécurité. En effet, ils soutiennent qu'il suffit d'introduire quelques dispositifs matériels, l'Élement Sécurisé par exemple, dans les architectures mobiles afin d'en garantir la sécurité. Cette vision nous semble simpliste et inadéquate pour plusieurs raisons.

Premièrement, un attaquant ayant un accès direct à l'ordiphone cible représente une vraie menace pour sa sécurité. En effet, n'importe quel dispositif matériel peut être analysé afin de mesurer des informations venant des différentes sources. Ces mesures peuvent être utilisées afin de mener des attaques de type canaux cachés ou auxiliaires. Dans le meilleur des cas, on utilise des dispositifs résistants à ce genre d'attaques en ne dévoilant aucune information utile.

Deuxièmement, un attaquant n'ayant pas cet accès direct pourrait encore compromettre la sécurité des logiciels exécutés dans le dispositif de confiance en exploitant, par exemple, des failles de type dépassement de tampon (en anglais buffer overflow). Idéalement, les logiciels exécutés ont été formellement vérifiés à la fois au niveau du code écrit et au niveau de sa représentation binaire générée par le compilateur.

Troisièmement, des micrologiciels malveillants ou des chevaux de Troie pourraient très bien être insérés lors de la phase de fabrication, y compris lorsque des vérifications formelles auraient été réalisées. Ainsi, notre confiance psychologique dans les dispositifs matériels est inversement proportionnelle à leur taille/complexité, ce qui implique que plus la taille de l'implémentation est grande, plus la probabilité d'avoir des vulnérabilités et des portes dérobées est importante.

3 Nos contributions

Alors que nous admettons que ce modèle de confiance basé uniquement sur les dispositifs matériels est encore valable pour les concepteurs des ordiphones qui se contentent de satisfaire leurs objectifs de court terme, dans cette thèse nous cherchons à apporter une compréhension plus fine de ce que signifie la confiance pour les ordiphones. À cette fin, nous approfondissons deux approches de confiance, à savoir la sécurité prouvée et l'approche à double environnement d'exécution.

Plus précisément, nous avons réalisé les contributions suivantes:

- **Une effraction dans l'entrepôt de clés du système Android.** En raison de sa grande popularité, nous nous sommes intéressés au système Android et avons étudié un des mécanismes de chiffrement utilisé par ce système avec les outils de la sécurité prouvée. Le schéma étudié est implémenté dans le composant définissant l'entrepôt de clés (en anglais KeyStore) du système Android. Ce composant protège la confidentialité et l'intégrité des clés secrètes stockées dans les ordiphones. Étant donnée sa place importante dans l'architecture d'Android, ce composant a dû recevoir une attention particulière pour assurer sa sécurité. Néanmoins, nous prouvons que le schéma utilisé ne garantit pas l'intégrité des données chiffrées, ce qui signifie que l'adversaire (l'attaquant dans la terminologie de la sécurité prouvée) est capable de modifier les clés stockées à l'insu des utilisateurs. Ensuite, nous continuons notre investigation en définissant un scénario où l'attaquant peut compromettre la sécurité garantie par l'entrepôt de clés. En particulier, une application malveillante peut rendre toute opération cryptographique vulnérable aux attaques exhaustives en réussissant à convaincre les autres applications d'utiliser des clés faibles.

- **Une cryptanalyse de la famille des protocoles sécurisés définis par GlobalPlatform.** Nous poursuivons nos analyses des systèmes déployés en examinant les cartes à puce. Nous faisons un tel choix pour deux raisons. La première est que la sécurité des cartes à puce est fortement liée à celle des ordiphones car de nombreuses architectures mobiles se basent sur les cartes à puce (communément appelées Éléments Sécurisés) afin d'assurer leurs propriétés de sécurité. La deuxième raison est que la sécurité des cartes à puce a été approuvée par des organismes de certification internationaux. Dans cette étude, nous nous intéressons à la famille de protocoles sécurisés SCP (de l'anglais Secure Channel Protocols) qui a été définie dans l'objectif de protéger la gestion à distance pour les applications des cartes à puce. Nous trouvons deux résultats intéressants grâce à notre analyse. D'une part, et malgré sa certification, nous démontrons une attaque théorique contre la confidentialité de SCP02 qui est le protocole le plus populaire de la famille SCP. L'attaque identifiée permet à une entité malveillante de retrouver les messages en clair correspondant aux chiffrés générés pendant la session sécurisée établie par SCP02. D'autre part, nous approfondissons l'analyse de sécurité de SCP03 qui, au moment de la rédaction de ce manuscrit, est le membre le plus jeune de la famille SCP. Nous prouvons que SCP03 satisfait plusieurs notions de sécurité en résistant aux attaques par rejeu, aux attaques par désordre dans la livraison des messages, et plus notamment aux attaques par substitution d'algorithmes.

- **Un nouvel éclairage sur l'approche à double environnement d'exécution.** Après avoir appliqué les outils de la sécurité prouvée sur des systèmes simples, nous intéressons à la confiance dans des architectures plus complexes. Nous commençons cette partie par un simple constat : les approches actuelles qui renforcent la sécurité des ordiphones ne sont pas suffisantes pour garantir une exécution sécurisée aux applications sensibles. En conséquence, nous revisitons l'approche à double environnement d'exécution dual-EE (de l'anglais Dual-Execution-Environment), considérant son potentiel prometteur en termes de sécurité pour les systèmes réels. Cette approche consiste à scinder le système en deux parties isolées. Chaque partie garantit un niveau de sécurité différent aux applications exécutées. En particulier, une partie exécute les composants de confiance, alors que l'autre partie exécute les autres composants, y compris le principal système d'exécution comme Android. Souffrant d'un manque de formalisation, nous explorons davantage cette approche. Ainsi, nous proposons un cadre précis afin de systématiser l'étude des solutions implémentant l'approche dual-EE en se référant à l'architecture MILS (de l'anglais Multiple Independent Levels of Security). Ce cadre défini nous permet de bien évaluer les technologies connexes, et donc de nous aider à identifier la technologie la plus pertinente à utiliser dans la suite de nos travaux. Ladite technologie est ARM TrustZone.

- **Un modèle abstrait pour l'environnement d'exécution de confiance.** Nous combinons la technologie ARM TrustZone avec l'approche dual-EE afin de construire ce que nous appelons *environnement d'exécution de confiance* ou TEE (de l'anglais Trusted Execution Environment). C'est GlobalPlatform qui a été le premier à utiliser ce terme. Le TEE est souvent défini en tant qu'un environnement d'exécution isolé capable d'assurer l'exécution sécurisée de ses applications. Malgré son déploiement à grande échelle, les définitions existantes de TEE sont inconsistantes et imprécises, ce qui induit une confusion dans l'utilisation de ce terme. Cette confusion a des conséquences négatives sur les conceptions finales proposées pour le TEE. En conséquence, et afin de mieux comprendre sa portée et ses limites, nous présentons une définition d'un TEE, que nous pensons être plus claire, en précisant bien les blocs composant son architecture. Ensuite, nous analysons cette architecture en détaillant ses propriétés et comment elles peuvent être atteintes. Enfin, nous présentons un modèle de sécurité propre au TEE avec les vecteurs d'attaques possibles contre les solutions industrielles implémentant le TEE.

- **Un TEE à base de *circuits embrouillés*[5].** Avec notre modèle de TEE, nous identifions une faiblesse récurrente dans les solutions de TEE proposées, à savoir la faiblesse de leur niveau de sécurité garanti. En effet, plusieurs attaques ont été implémentées contre des solutions de TEE déployées à grande échelle. Ces attaques mettent en cause la promesse de cette technologie d'offrir une meilleure sécurité. Quand on le compare à l'Élément Sécurisé, le TEE est perçu comme un compromis offrant davantage de puissance de calcul, mais garantissant moins de sécurité. C'est la raison pour laquelle nous explorons une nouvelle architecture de TEE qui permet de protéger les données sensibles même quand le cœur du TEE a été compromis. À cette fin, nous utilisons des techniques particulières *issues du calcul réparti sûr à deux parties* (en anglais Secure Two-Party Computation). Notre objectif est de redéfinir le modèle d'exécution de TEE afin d'affaiblir les exigences en termes de confiance dans l'écosystème de TEE. Ainsi, les fournisseurs des services, tels que les opérateurs mobiles et les banques, pourront profiter à la fois de la sécurité garantie par le TEE et de sa puissance de calcul sans être contraints de faire pleinement confiance aux fournisseurs de TEE.

- **Un TEE pour les systèmes autonomes.** Notre dernière contribution dans ce manuscrit est consacrée à l'intégration du TEE dans un domaine différent mais proche des ordiphones, à savoir les systèmes autonomes. Les systèmes autonomes sont ceux qui sont conçus dans l'objectif de minimiser leurs besoins en termes de gestion humaine. Ils assurent plusieurs propriétés, notamment l'auto-protection et l'auto-remédiation. Étant sensibles, ces deux propriétés ont besoin d'un certain niveau de sécurité pour être garanties. Pour cela, nous nous basons sur les propriétés d'isolation du TEE afin d'avoir une telle garantie. Notre nouvelle solution a été construite afin de permettre aux systèmes autonomes de protéger leur système d'exploitation contre des attaquants puissants. En outre, grâce à notre solution, nous assurons que le système retrouve son état normal dans le cas où un attaquant aurait réussi. Ce travail permet d'élargir le spectre d'applications du TEE en montrant qu'il peut offrir des propriétés de sécurité plus élaborées à des composants assez complexes comme un noyau d'un système d'exploitation. Illustré dans le cadre des systèmes autonomes, ce travail est applicable à l'architecture des ordiphones.

Nous estimons que ces contributions permettent d'approfondir notre compréhension dans le domaine de l'informatique de confiance appliquée aux ordiphones, et nous espérons qu'elles permettront de faire avancer les travaux scientifiques s'intéressant à ce domaine particulier. En effet, nous établissons d'abord le grand intérêt de la sécurité prouvée dans l'analyse des schémas cryptographiques implémentés dans des systèmes déjà déployés. Malgré le fait qu'elle soit perçue trop souvent comme un outil théorique, nous montrons qu'elle peut être habilement utilisée pour trouver des failles de sécurité théoriques qui pourraient avoir des conséquences pratiques plus sérieuses pour les systèmes concernés. Ainsi, nos résultats permettent de voir comment la sécurité prouvée peut servir à la fois à identifier des vulnérabilités existantes ou au contraire garantir leur absence quand le système est prouvé sûr. Cela souligne la polyvalence d'un tel outil, et invalide l'assertion selon laquelle la sécurité prouvée ne pourrait être utilisée qu'à des fins théoriques. Deuxièmement, nous pensons que nos travaux démontrent également le grand potentiel du TEE. En effet, GlobalPlatform, qui est l'organisme en charge de la publication des spécifications industrielles concernant le TEE, est un consortium composé principalement d'entreprises travaillant dans le monde de la carte à puce. C'est la raison pour laquelle le TEE de GlobalPlatform a été conçu à l'image de la carte à puce à tel point qu'une partie de leurs spécifications suivent les mêmes principes. Notre objectif est de dépasser cette vision restreinte en montrant comment le TEE nous permet de résoudre des problèmes de sécurité intéressants quand on reconsidère son architecture.

[5] Garbled Circuits

Il convient de signaler la compétitivité et la rapidité de l'évolution du marché des ordiphones. Ce constat implique que les nouvelles solutions de sécurité destinées à ce marché devront être facilement déployables, et avoir un bon rapport coût/efficacité. Ainsi, nous concevons nos solutions pour qu'elles ne soient pas seulement capables de résoudre des nouveaux problèmes de sécurité, mais aussi pour qu'elles soient conformes aux exigences du marché. En effet, notre travail sur la sécurité prouvée démontre qu'une intégration de cet outil au cycle du développement des composants sensibles permet d'éviter la complexité de leur mise à jour. Quant à la deuxième partie, ce sont les aspects pratiques et la facilité de déploiement qui sont au cœur de notre approche de dual-EE dans cette partie. En outre, l'objectif principal de la construction d'un TEE à base de circuits embrouillés est de simplifier les contraintes liant ensemble les différents acteurs dans l'écosystème du TEE, et de ce fait diminuer le coût et la complexité du déploiement.

4 Structure du manuscrit

Ce manuscrit est divisé en trois parties. La première partie a pour objectif de bien définir le contexte de nos travaux de recherche. Le chapitre 2 introduit les prérequis techniques pour la bonne compréhension de ce manuscrit. En outre, nous détaillons les défis concernant la sécurité des ordiphones ainsi que les principaux concepts connexes. Au chapitre 3, nous introduisons le cadre formel de la sécurité prouvée que nous allons utiliser dans la suite de ce manuscrit. Nous présentons aussi les modèles de sécurité relatifs à nos travaux, notamment ceux des schémas de chiffrement symétrique.

La deuxième partie concerne nos contributions obtenues en appliquant les outils de la sécurité prouvée dans le contexte des ordiphones. Au chapitre 4, nous étudions l'entrepôt de clés, qui est un composant d'une grande importance pour le système d'Android. Ce chapitre identifie une faille de sécurité que nous avons communiquée à l'équipe de sécurité d'Android et qui concerne une vulnérabilité dans le schéma cryptographique utilisé pour garantir l'intégrité et la confidentialité des données stockées. Nous continuons cette partie en menant au chapitre 5 une analyse formelle de la famille des protocoles sécurisés sur lesquels s'appuient les cartes à puce pour protéger la gestion à distance de leur contenu.

Concernant la troisième partie, elle est consacrée à nos contributions liées à l'environnement d'exécution de confiance TEE. Nous débutons cette partie en proposant un fondement théorique de cette technologie au chapitre 6 qui revisite l'approche à double environnement d'exécution. Nous présentons notre propre définition et modèle de TEE au chapitre 7. Nous présentons dans le chapitre 8 les avantages de notre modèle. Nous concevons en effet deux architectures de TEE dans ce chapitre: (1) une architecture de TEE basée sur les circuits embrouillés qui permet de simplifier le modèle de confiance pour le TEE, et (2) une solution utilisant les propriétés d'isolation du TEE afin de protéger le composant assurant l'auto-protection et l'auto-remédiation dans les systèmes autonomes.

Le chapitre 9 conclut ce manuscrit en présentant un récapitulatif des résultats obtenus et les perspectives pour des futurs travaux.

Chapter 1

Introduction

"The White Rabbit put on his spectacles,
Where shall I begin, please your Majesty? he asked.
Begin at the beginning, the King said, very gravely,
and go on till you come to the end: then stop."

– Lewis Carroll, Alice's Adventures in Wonderland

Contents

1.1 Motivation **7**
1.2 Research Topic **8**
1.3 Contributions **9**
1.4 Outline **11**

In this chapter, we briefly present our work in the field of mobile security and how we contributed to enhance the security of mobile devices, in particular smartphones. We begin by motivating the need for mobile security and by pinpointing where problems surface. Thereafter, we provide details about our contributions and the specific fields on which we work. We then close this introduction with an outline of the remainder of this dissertation.

1.1 Motivation

The first smartphones were introduced in 2007 by Apple followed by Google in 2008, and since then smartphones have become almost vital in the modern world. According to recent statistics [IDC16b], the worldwide market of smartphones has reached a total of 1.48 billion units in 2016, 82.5% of which runs on Android. Smartphones owe their success to their size that fits into pockets, to their great computational power that is enough to smoothly execute a large set of applications, and to their ubiquitous connectivity that allows users to stay online all the time. These three features make smartphones great tools that help us accomplishing our daily-life tasks.

The popularity of smartphones is also caused by vendors that make sure that users have easy access to their services via mobile applications. These applications can be installed in very few steps through so called *App stores*. The amount of available applications is astonishing: more than 2 million applications were available in February 2016 [Sta16]. While some applications must be purchased, the majority can be installed and used free of charge, which also contributes to the smartphone's popularity.

Beside the basic communication services, these devices allow people to stay connected on their smartphones sending messages with their friends, exchanging confidential emails with their coworkers, managing their bank account, and even buying arbitrary things from real shops. Many of the performed activities on smartphones (in)directly require to store sensitive data, such as

users private credentials or secret company data. This raises important security questions, as the connected and the open nature of smartphones results in a huge potential for malware.

Indeed, connectivity as well as the vast amount of installable applications are very appealing to evildoers. For instance, they can generate money if they can steal payment information stored on the victim's smartphone, such as credit card numbers. Stealing account credentials is also attractive, as these accounts can be then used to perform further attacks. Furthermore, acquired confidential emails can be used to discredit involved companies.

These examples are a very short excerpt of what can "go wrong" on smartphones. Indeed, since its first appearance in 2008, More than 540 security vulnerabilities have been reported against Android on CVE by September 2016 [CVE16a]. Being increasingly aware of the security of their devices, users are still reluctant to adopt smartphones to execute sensitive applications, since they do not *trust* smartphones to protect them from malware or other security issues. This hinders big service providers for smartphones like Orange in their efforts of inventing new innovative applications that require security for their sound execution [Bro16]. Problems like these have led us to the huge field of mobile security.

1.2 Research Topic

Smartphones are increasingly becoming an enticing target for evildoers who are seeking to widen their attacks on smartphones for fun and profit. Therefore, malicious applications for smartphones are becoming more sophisticated and larger by their numbers. Although academic and industrial research provides a variety of useful mechanisms and tools to face these growing security challenges, they are only partially effective in practice as long as smartphone architectures suffer from several (conceptual) security problems of software (especially their running operating systems) and underlying hardware. To reach the desirable short and long term security goals, the smartphone industry needs a cost-effective, safe and careful redesign of their architectures to embed security features by design based on reasonable trust assumptions.

Thus, there is a trend to integrate *Trusted Computing* concepts inside smartphones. Trusted Computing is a special branch of computer security. As specified by Gasser [Gas88], a trusted software is a one that "is responsible for enforcing security, and consequently the security of the system depends on its flawless operation". Since its advent, Trusted Computing has raised big debates within the security community. Indeed, the terminology 'Trusted Computing' is quite unfortunate. It implies that some party trusts the software in question. This assertion says nothing about who that party is, whether the software is worthy of that party's trust, and on what basis that party chooses to trust it. The main problem is that trust is a subjective property, hence non-measurable. In English, according to the Cambridge dictionary, trust is the "belief in honesty and goodness of a person or a thing". A belief is hard to capture in a quantified way. The notion of trust is more subtle in the field of computer systems. In the real world, an entity is trusted if it has behaved and/will behave as expected. In the computer field, we can plausibly assume that trust follows the same assumption.

Trusting all the software in a large system is hopeless; hence, systems are structured in a way minimizing the set of components needing trust. Thus, within a single system, it is normally useful to distinguish between trusted components and untrusted ones. A trusted component is a one that does enforce security and does not have security flaws, while an untrusted component is a one that does have security flaws or attempts to actively subvert the system. This interpretation is especially common in environments where extremely sensitive information is handled, and it constitutes a fundamental tenet to build a secure system. In an ideal world, trusted components are those that first have been developed by trusted individuals according to strict standards, and second have been demonstrated to be correct by means of advanced techniques, such as formal modeling and formal verification. This standard for trust far exceeds the standards applied to

most existing computing systems, since they are considerably complex and costly to implement.

Some may simplify their model by reducing security to their idealized trust in hardware (HW). Indeed, they believe that it is enough to integrate hardware primitives, such as Secure Elements, into their architectures in order to guarantee security. We argue that this idealized definition of a trusted HW device is unsatisfactory for multiple reasons.

First, an attacker with physical access represents a real threat for HW security. Indeed, a HW device can be opened up and looked inside in order to perform some side channel measurements. As a matter of fact, analogue measurements from observations can always be used to extract some information at some positive rate. In addition, intermediate computed values can often be explicitly targeted. At best, leakage resilient devices are used to protect such assumed attackers.

Second, a remote attacker (i.e. without direct physical access) may try to compromise the software running on the trusted HW device by, for instance, exploiting a buffer overflow vulnerability. At best, the higher level language in which the software was written has been formally verified and the used compilers are proven to correctly translate the code to binaries.

Third, even if formal verification was performed, malicious firmware or HW Trojans may be inserted. Thus, our psychological trust in HW seems inversely proportional to its size/complexity: the larger the size of the HW implementations, the larger the probability of having vulnerabilities, Trojans and side channels that can be exploited.

1.3 Contributions

While we acknowledge that such a trust model may still be valuable to mobile device manufacturers and mobile platform providers in the short term, our intention in this thesis is to provide a fine-grained understanding of what is meant by trust in smartphones. To this end, we work on two trust approaches: the provable security and the dual-execution-environment approaches.

More precisely, we make the following contributions:

- **Breaking into the Android KeyStore.** The first system on which we apply the paradigm of provable security is the Android KeyStore. The Android KeyStore is the component that shields users cryptographic keys when stored on mobile devices. Being of paramount importance, the KeyStore is one of the key components in Android, and therefore much engineering effort has been made to ensure its security. Nevertheless, we prove that it does not withstand a simple analysis under the paradigm of provable security. Indeed, we formally prove that the encryption scheme used by the KeyStore does not provide integrity, which means that an adversary (an attacker in the provable security language) can undetectably modify the stored keys. Then, we push our investigation further and define a concrete scenario breaching the security guaranteed by the KeyStore. In particular, the scenario allows a malicious application to make mobile apps to use weak keys, thereby making all subsequent operations involving cryptographic schemes vulnerable to exhaustive attacks.

- **Cryptanalysis of GlobalPlatform Secure Channel Protocols.** Being surprised by our findings concerning the Android KeyStore, we apply the mechanisms of provable security on another system that is highly approved by an international certification institution, namely the family of Secure Channel Protocols (SCPs) that represents the cryptographic heart of the system of content management in the sensitive industry of smart cards. The security of smart cards is strongly related to that of smartphones due to the fact that many mobile architectures rely on smart cards (often called Secure Elements) to ensure their security. We provide two results in our analysis. On the one hand, we demonstrate an attack against the confidentiality of SCP02, which is the most popular protocol in the SCP family. The identified attack allows a malicious entity to recover encrypted messages during a secure session established by SCP02. On the other hand, we investigate the security of SCP03

which is, at the writing of this dissertation, the most recent SCP member that is based on symmetric keys. We find that SCP03 provably satisfies strong notions of security. More notably, it withstands replay, out-of-delivery and algorithm-substitution attacks.

- **Revisiting the Dual-Execution-Environment Approach.** After applying provable security on quite simple systems, we then look at the security of more complex mobile systems. We start our study by a simple observation: the current approaches of enforcing mobile security are not enough to offer a secure execution environment for sensitive applications. Therefore, we revisit the Dual-Execution-Environment (dual-EE) approach due to its identified promising potential in terms of providing strong usable security. This approach consists of splitting the mobile system into two strongly isolated subsystems, one of which is trustworthy and the other one runs the full-fledged mobile operating system. Not yet well formalized, we investigate this approach, and thus propose a framework to systematize the design of dual-EE solutions regarding the architecture of the Multiple Independent Levels of Security (MILS). This framework allows us to better compare the existing dual-EE technologies, thereby helping us identifying the best technology to use.

- **Abstract Model for Trusted Execution Environments.** The aforementioned technology is ARM TrustZone that we combine together with the dual-EE approach in order to build the so called *Trusted Execution Environment (TEE)* coined by GlobalPlatform. TEE is commonly known as an isolated processing environment in which applications can be securely executed irrespective of the rest of the system. Despite its wide deployment, existing definitions of TEE are largely inconsistent and unspecific, which leads to confusion in the use of the term and its differentiation from related concepts. Therefore, and in order to better understand the extent of this technology, we proceed by proposing a precise definition of TEE. Furthermore, we extensively analyzed its core properties by identifying the most relevant approaches to achieve them. This abstract model allows us to reconsider the power of TEE, and hence broaden its scope in the security domain.

- **Trusted Execution Based on Garbled Circuits.** One of the recurrent weaknesses in TEE solutions is the security level guaranteed by the TEE. Indeed, several attacks have been recently disclosed against deployed TEE systems, which jeopardize the security promise of such a technology. When compared to the tamper-resistant Secure Element, TEE is often perceived as a tradeoff between more computation power, but less security. Thus, we explore a new design that allows TEE to protect its sensitive data even when the secure kernel gets compromised. We achieve this by using efficient techniques for two-party secure function evaluation. The main interest of our architecture is to weaken the *trust requirement* for the TEE. Thus, Service Providers, like Mobile Operators or Banks, can benefit from the TEE security and power without having to fully trust the TEE issuer.

- **Trusted Execution for Autonomic systems.** We end up this dissertation by applying the TEE principles in another context that is different from smartphones, namely embedded autonomic systems. Autonomic systems are those that ensure several properties minimizing human intervention. In particular, autonomic systems ensure self-protection and self-healing. These two properties require a high level of security to be guaranteed. To this end, we rely on the TEE strong isolation to provide such a guarantee. Our new solution is designed to ensure protection and graceful remediation of the main OS kernel against powerful attackers. This work shows that TEE can be used not only to protect the execution of sensitive applications, but also to provide more elaborated security properties (i.e. self-protection and self-healing) to a complex software system like the OS kernel.

We argue that these contributions enrich the research work by opening new directions for reflection on trust for smartphones. On the one hand, we show the great interest of analyzing

real-world systems using the paradigm of provable security. Despite being a theoretical tool, we show that it can be handily used in order to find theoretical security flaws that could lead to serious consequences for the related system. Thus, by taking the overall of our findings, we demonstrate how provable security can be used to both strongly guarantee the absence of security vulnerabilities and identify them if they exist. This shows the versatility of such a tool, and dismisses the idea that provable security can be only used to study the security of theoretical systems. On the other hand, we show the untapped potential of TEE. Indeed, GlobalPlatform, which is the institution that publishes extensive specifications for TEE, is a consortium that is mainly composed of companies related to the industry of smart cards. This is why TEE was designed in the image of smart cards, and many of their respective specifications follow the same principles. Our work goes beyond this. It aims at proving that the true power of TEE is not only that it offers more computation capacities to perform resource-demanding secure applications, but also that it can be better integrated into the main system of smartphones to offer better security when combined with the appropriate tools. Thus, we hope that this thesis constitutes an attempt to fill in the gap in our understanding of trust in the security of smartphones.

It is worth noting that there is nothing as fast-paced and competitive as the smartphone market. Thus, new security solutions require to be cost-effective and easy to implement in order to have a chance of being deployed into real devices. We take great care when designing our architectures, so that they both solve challenging security problems and meet the imperatives of the industry of smartphones. Indeed, our work on provable security shows that some kind of integration of this tool into the development of some sensitive components helps avoid posterior security updates that might be costly or hard. As for the second part, usability and ease of deployment are in the core of our model of the dual-EE approach. In addition, the main goal of designing a new TEE architecture based on Garbled Circuits is to simplify the constraints binding the different actors of the TEE ecosystem, thereby decreasing its deployment complexity/cost.

1.4 Outline

This dissertation is split into three parts. The first part sets the context of our research. Chapter 2 introduces the required technical background for the comprehension of this thesis. We also discuss the difficulties challenging the ecosystem of smartphones as well as the core concepts of smartphone security. In chapter 3, we introduce the framework of provable security as well as its related concepts. We also provide the security models that will be used in later chapters.

The second part is about our contributions related to applying provable security in the context of smartphones. In chapter 4, we study the KeyStore, which is an important security component in the Android architecture. This chapter shows a security flaw that was acknowledged by the Android security team and that concerns the encryption scheme used by the KeyStore to protect its data. We proceed our study in chapter 5 where we analyze the family of Secure Channel Protocols on which smart cards rely to secure their content management.

As for the third part, it consists of our contributions related to the Trusted Execution Environment (TEE). We start this part by setting up a sound theoretical foundation of this technology in chapter 6 that revisits the Dual-Execution-Environment approach. We present our own definition of TEE in chapter 7 by distilling its different core components. Allowing us to go beyond the current TEE limitations, the benefits of our model are presented in chapter 8 in which we introduce two advanced architectures: (1) a TEE design based on Garbled Circuits that helps improves the trust and security model of TEE, and (2) a solution integrating the TEE into autonomic systems in order to ensure protection and graceful remediation of the main OS.

We end this dissertation by chapter 9 that draws some concluding remarks, summarizes our major contributions and discusses perspectives, challenges and future work directions.

Part I

Preliminaries

Chapter 2

Trust on Smartphones

"Never trust anything that can think for itself if you can't see where it keeps its brain."

– Joanne K. Rowling, Harry Potter and the Chamber of Secrets

Contents

2.1 Important Note **15**
2.2 Smartphone Security **16**
2.2.1 Security Model 16
2.2.2 Security Challenges 18
2.3 Security Internals of Android **20**
2.3.1 Android Architecture 20
2.3.2 Security Enforcement 21
2.3.3 The Binder 22
2.3.4 The Android KeyStore 22
2.4 Secure Hardware Technologies in Smartphones **24**
2.4.1 The Baseband 24
2.4.2 The Secure Element 25
2.4.3 ARM TrustZone 27
2.5 Conclusion **29**

This chapter introduces the required technical background necessary for the comprehension of this thesis. Firstly, we begin by a detailed discussion of the difficulties challenging the ecosystem of smartphones as well as an introduction to the core concepts of smartphone security. Secondly, we dip into the internals of the most security-critical components of Android. In the last part of this chapter, we introduce several hardware primitives that are used in later chapters to build trusted solutions for smartphones. In particular, we present the Baseband, the Secure Element (SE) and ARM TrustZone.

2.1 Important Note

This chapter does not pretend to cover the topic of *smartphone security* in its global generality. Nowadays, smartphones come in various flavors. They run on different operating systems and are being produced by many manufacturers. Each operating system has its own security architecture. Although, it might be of independent interest to analyze the security architecture of all the existing smartphone operating systems, this would deviate the reader from grasping the main contributions

of our thesis. Indeed, our solutions would have been cluttered with too many technical details about the particularity of each operating system. Therefore, we decide to focus solely on one single smartphone architecture, but at the same time we argue that our contributions are not limited to that architecture and can be easily adapted to other ones.

In this thesis, we choose *Android* as our working smartphone environment. According to IDC (International Data Corporation) [IDC16a], Android market share in $Q2$ 2015 was 82.8 percent of the worldwide market (as calculated by shipment volume) with 283 million units shipped. Apple's iOS had 13.9 percent of the market in the same quarter, Windows Phone and BlackBerry followed behind with only 2.6 percent and 0.3 percent respectively. Since $Q2$ 2013, Android represents more than 80 percent of the market. With that much market share, coupled with the interesting research happening in the Android world, we feel that considering Android gives more value to our work than if we had chosen another architecture.

2.2 Smartphone Security

For researchers working in the domain of information security, the smartphone space represent a fairly new and sparsely charted continent to explore. Indeed, it includes diverse geography in the form of different processor architectures, operating systems, hardware peripherals and software stacks. All of these create an ecosystem for a diverse set of vulnerabilities to exploit and study.

Therefore, we begin this section by presenting the security model of smartphones, and then we explain why they are peculiarly hard to secure.

2.2.1 Security Model

Security is one of these concepts in computer science that carry multiple meanings. In this thesis, we define a secure system as it is stated in [Jae08] as "*system that includes security mechanisms which ensure that the system's security goals are enforced despite the threats faced by the system*".

According to this definition, security is a three-dimensional concept that requires three aspects to be taken into account: security goals, threat model and security mechanisms. In follows, we discuss the two first aspects here and we defer the discussion of security mechanisms to section 2.3.

Security Goals

The security goals are the requirements which a secure system must satisfy against specific threats. The security of smartphones deals with the protection of the running system and the data stored on the device. It aims at guarding smartphones from unauthorized access, disruption or destruction. Consequently, as for many other systems, there are main three security goals for smartphones: confidentiality, integrity and availability (CIA) [ISO16]. The task of smartphone designers is to construct a security policy that finds the right balance among these CIA triad. A *security policy* is a set of rules describing the security specifications for the related system. These security specifications should be well-defined, implementable and unambiguously expressed.

Confidentiality. Confidentiality is about keeping the stored data from being disclosed into unauthorized hands. An example of this would be protecting the necessary data for a credit card transaction from being intercepted by a third party. A secure smartphone ensures the confidentiality of its sensitive data, otherwise significant financial or personal losses might be endured by users.

Integrity. Integrity is about ensuring that any unauthorized modification to data will be detected. This guarantees that all data accessed by an authorized user has not been tampered with, and therefore they are trustworthy. The absence of integrity may result in catastrophic

damage. For instance, a credit card transaction can be easily compromised if attackers can maliciously change its corresponding data.

Availability. In order for any system to be of use to its users, data should be available when they are required. This is the principle of availability, which means that smartphones must keep their system working correctly and ready for any interaction with their user.

Threat Model

A threat model is an external event that can damage or comprise the system. It defines the set of threats which an attacker can use to exploit the system vulnerabilities and thereby breaching the security goals. Below, we discuss in details the smartphone attack vectors, which can be physical or logical:

Logical Attack Vectors.

- *Operating System*: applications may abuse potential weakness, and thus circumvent the security measures enforced by the operating system [ZDH+13].
- *Communication Services*: In contrast to emails, SMS and MMS are not checked by company firewalls, and thus vulnerable smartphones could be attacked anywhere by malicious SMS or MMS.
- *Browser*: In addition to the existing browser vulnerabilities [Woo10], smartphones offer further targets due to the interaction between the browser and the device. For example, the user identity connected to the SIM card might be abused.
- *Users*: Very often, users become accomplices in some kind of attacks because they were deceived into carrying out some critical actions. Moreover, users might disable automatic updates or poorly configure their device, which may promote attacks against the undefended devices [MBOS16].
- *Third Party Applications*: Beyond the danger of installing some malware on the smartphone [Dom11], applications also present a threat due to their own inherent vulnerabilities. Past experiences show that vulnerable applications may affect more than their own data.

Physical Attack Vectors.

- *Firmware*: The firmware integrity represents the basis for many security functions. Maliciously manipulated firmware may allow attackers to obtain complete remote control over the smartphone and its data [GRS13].
- *Memory*: A direct access to the contents of the computing memory (i.e. RAM) often offers an opportunity to overwrite protective mechanisms or just read user data.
- *Hardware Interfaces*: An attacker may use memory buses and hardware interfaces (JTAG) to circumvent protection mechanisms of smartphones.
- *SIM*: Even though the SIM card itself, as we will see later, is highly secure, attackers might be able to manipulate the communication interface between the SIM card and the smartphone in order to read some critical data and possibly change it [XN15].
- *Baseband Processor*: Home-crafted base stations are quite inexpensive. Attackers can use these base stations to compromise the mobile radio interfaces [MM09]. Moreover, rogue radio interfaces allow attackers to breach the infrastructure of the network service providers.

- *Wireless Interfaces*: Attackers can take advantage of the vulnerabilities in radio communication interfaces (Bluetooth, NFC, Wifi, etc.) in order to illegally obtain user data [AF12].

- *Memory Cards*: Data on external media is generally left unprotected. Thus, an attacker might be able to easily read the data if she has a direct access to the smartphone. In addition, manipulated data could be stored on the memory card in order to infect the smartphone system later.

2.2.2 Security Challenges

Conflicting Traditions

The landscape of mobile devices has been changed with the introduction of smartphones. Until 2007, mobile devices are used only for making voice calling and text messages. The old class of mobile devices, referred to as feature phones, provides basic Internet and multimedia functionalities. Feature phones have a limited operating system that is owned and controlled by the device manufacturers or cellular carriers. Such a control was crucial for cellular carriers, since their security model was based on the assumption that the baseband firmware is non-malicious. Cellular carriers defend their networks by first enforcing the certification of the baseband firmware [Com16a], and then closing the whole system. Thus, no security update was necessary.

In contrast, the new class of mobile devices, namely the smartphones, abandoned the close-system model. Instead, they run a full fledged operating system and allow users to install third-party applications. The great benefit of such openness is more flexibility for users who are able to freely install various software. This does not come without risk. Open, not to mention connected, smartphones are exposed to remote attacks. It is worth noting that smartphones are also mobile devices, and therefore they include a cellular environment. A security challenge in smartphone design is to cope with this blend of conflicting environment. The conflict between the restrictive cellular environment with the open operating system has caused a number of limitations in the smartphone design. For instance, the operating systems of smartphones limit the access to some parts of the system, disallowing users from modifying some components. Even "open" architecture like Android does not allow users to become the true master of their devices [HSG11]. This is because in the cellular tradition access to the radio interface is prohibited.

An indirect consequence of such a blend is the complexity of the update process of smartphones. Indeed, the limited operating system in feature phones did not need to be patched as regularly and as quickly as in smartphones. However, in the world of general purpose computing, quick update is deeply integrated, since designers understand that vulnerabilities exist, and once they are discovered, the system needs to be patched. For instance, Microsoft releases its security patches on every second Tuesday of the month. Unfortunately, smartphones inherit the worst of both worlds. Their operating systems are as complex as those of personal computers, and therefore vulnerabilities exist and patches are required. Nevertheless, the patch process is long and complex [FM06, Zor16]. The slowness of the patching process implies that vulnerabilities remain unfixed for ages.

The Trinity of Trouble

Why are smartphones so difficult to secure? The Trinity of Trouble–connectivity, extensibility and complexity– allows us to answer this question [McG06].

Complexity. It is well known that the density of security flaws is directly proportional to square of the number of the lines of code [Hat97]. Modern operating systems of smartphones are comprised of at least millions of lines of code. Implementing such huge systems with zero flaw is

practically impossible. It is important to note that complexity in smartphones has more impact, since post-deployment fixes like patches are not mature.

Extensibility. modern smartphones are extensible. An extensible system means that its functionality can be modified via loadable modules and code. Most smartphones support extensibility through their application market where users can install third-party applications on their devices. Extensible systems are very attractive because they provide flexibility. However, at the same time, extensibility has the ease with which attackers trick users into installing malicious software without their knowledge or approval.

Connectivity. A third factor negatively affecting smartphones security is their connectivity to the Internet. This ubiquity of Internet connection puts smartphones at great risk because the network is the primary vector to exploit system vulnerabilities. Thus, connectivity has spurred the propagation of security breaches. Indeed, a physical access to smartphones is not needed to exploit poor coding or design problems.

Smartphone Security Versus Personal Computer Security

Despite the similarities between smartphones and personal computers, there are several notable differences concerning security. Firstly, malware can generate money from their illicit activities more easily on smartphones than in computers (premium rate calls/SMS are classified as the second most common behavior found in smartphone malware [FFC+11]). Secondly, Botha et al. in [BFC09] explore users point of view in a smartphone environment. They examine the availability of security mechanisms from users perspective. Indeed, authors have noticed that users generally wish to find their favorite computer-based security mechanisms in their smartphones, but many of these solutions are not applicable in smartphone environments. For instance, complex malware detection algorithms running on computers cannot be easily transferred to smartphones due to their limited computational power.

Compared to computers, the basic security principles of smartphones are quite different. The security problem of smartphones is particularly complex because of four reasons [Mul06]:

1. *mobility*: smartphones accompany their users wherever they go. Therefore, they can be easily stolen or physically tampered.

2. *strong personalization*: very often, smartphones are personal and have one unique user. Such strong personalization leads users to store different valuable data on their devices, which makes them more attractive to compromise.

3. *technology divergence*: smartphones are built upon various technologies designed by different manufacturers. This increases the attack surface of smartphones.

4. *reduced capabilities*: smartphones have smaller computational power compared to personal computers. It is true that this might limit the power of malware, but in the context of smartphones it especially hinders the adoption of malware that generally need to run resource-demanding algorithms.

A unique characteristic of smartphones is the battery, which severely limits the resources available for a security solution. Indeed, there must be a tradeoff between security and usability in a way that security solutions do not shorten the battery life [BFH+11]. This means that traditional approaches for malware detection might not be feasible for smartphones, since they consume a significant amount of resources and power. Moreover, a further security threat for smartphones stems from the fact that threats to user privacy in smartphones are quite different. Indeed, sensors are not optional and can be used to sniff users private data.

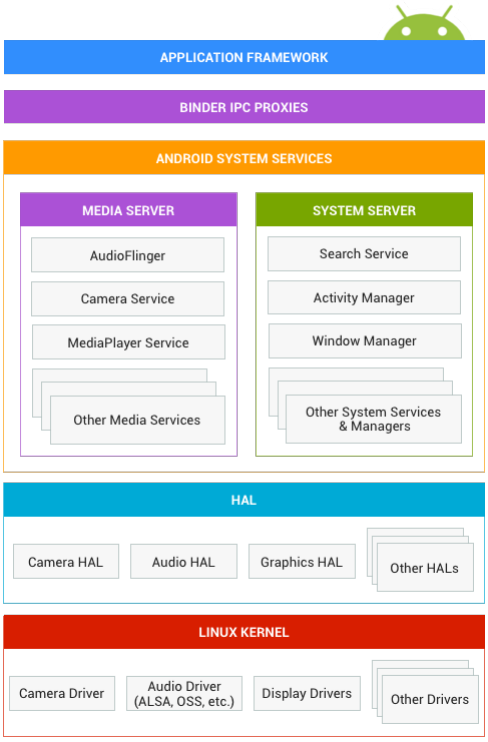


Figure 2.1: Android System Architecture [And16a]

2.3 Security Internals of Android

2.3.1 Android Architecture

Android is an open-source, Linux-based mobile operating system from the Open Handset Alliance, which is led by Google. Android applications are written in Java and compiled to Dalvik bytecode (.dex) that is a Java bytecode format designed for Android. In addition to Java code, an application may contain native libraries through the Java Native Interface (JNI). As depicted in Figure 2.1, the Android architecture consists of five main layers:

1. *Application Framework*: it provides developers with a rich API for the device functionalities. This includes building blocks to enable developers to perform common tasks, such as managing user interface (UI) elements, accessing shared data and passing messages between application components.

2. *Binder*: it provides Inter-Process Communication (IPC) between the application framework and lower layers. This enables high level framework APIs to interact with the Android system services.

3. *Native System Services*: they allow the application framework to have access to the underlying hardware.

4. *Hardware Abstraction Layer (HAL)*: it defines a standard interface for hardware vendors to follow, and therefore makes them free to provide their own implementation without affecting or modifying the higher level system.

5. *Linux Kernel*: Android is based on a special version of the Linux kernel with some modifications necessary for the mobile environment.

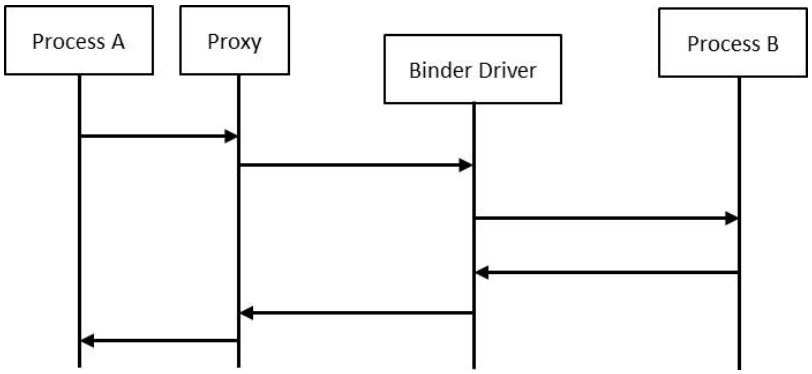


Figure 2.2: Binder Communication

2.3.2 Security Enforcement

Android deploys various security measures. Two main measures are sandboxing and the Android permission model. The former provides isolation for applications by taking advantage of Linux access control and process protection mechanisms. Based on ART (Android RunTime) virtual machine, the latter restricts applications capability by limiting the access to sensitive APIs. Users must give permissions for applications to use certain resources either during installation or runtime. Other deployed security measures include application signing to verify that different applications come from the same developer. A widely cited paper by Enck et al. [EOM09] gives an overview of Android and its security aspects. A thorough discussion about the security measures of Android is given by Shabtai et al. in [SFK+10]. Misra et al. [MD13] and Drake et al. [DLM+14] also cover various aspects of Android. Below, we briefly discuss Sandboxing and Android permissions.

Android Sandboxing

Android inherits process isolation and the principle of least privilege from Linux kernel. In particular, processes cannot interfere with each others as they run as separate users. Much of Android sandbox is defined from a few key concepts: standard Linux process isolation, unique IDs (UIDs) for most processes, and restricted file system permissions. Android shares Linux user ID/group ID (UID/GID) paradigm, but it does not have the traditional `passwd` and `group` files. Instead, it defines a mapping to unique identifiers known as Android IDs (AIDs). The initial AID mapping contains reserved static entries for privileged users. In addition to AIDs, Android uses supplementary groups to enable processes to access shared or protected resources.

The Android sandbox works as follows. When applications execute, their UID, GID and supplementary groups are assigned to a newly created process. Running under a unique UID and GID enables the operating system to enforce low-level restrictions in the kernel level.

Android Permissions

The Android permissions model is a three-faceted one:

- *API Permissions*: they include those that are used for controlling access to some functionalities within the Android application framework and, in some cases, third-party applications.
- *File System Permissions*: by default, applications have access only to their respective data storage paths on the file system. This is due to the fact that applications possess unique UIDs/GIDs, and only processes having those UIDs/GIDs could access the contents of their files/folders.

- *Inter-Process Communication (IPC) Permissions*: they relate to communication between applications.

After this short introduction to the Android security, we now focus on the two components that we will require later in this thesis: the Binder and the KeyStore.

2.3.3 The Binder

The Binder is an IPC mechanism that was added to the Linux kernel. In a nutshell, the Binder architecture operates in a client-server model. It allows a process to synchronously call methods in "remote" processes. The Binder architecture is defined in such a way that makes method calls seem as though they were local function calls. Figure 2.2 depicts the Binder communication flow.

At a high level, the exposed IPC methods are defined using an abstract interface via Android Interface Definition Language (AIDL). AIDL allows two applications to agree upon a specific interface for sending and receiving data. This keeps the communication interface separate from the implementation. In some way, AIDL is akin to C/C++ header files.

2.3.4 The Android KeyStore

The Android KeyStore is a high-level service in the application framework layer that enables applications to store their credentials. The original credential store was created in Android 1.6 and was limited to store VPN and Wi-Fi EAP credentials. Back then, only the operating system, and not user applications, could access the stored keys and certificates.

As illustrated in Figure 2.3, the KeyStore is comprised of three components:

1. Public APIs: provided by the Android Keystore provider and the KeyChain through Java class;
2. Keystore service: Binder-based system service that stores keys in encrypted form and mitigates unauthorized use of keys;
3. Keymaster: meant to decouple the KeyStore from its implementation.

Public APIs

The KeyStore services are accessible via the KeyChain API, introduced in Android 4.0, as well as the Android Keystore provider that was lately introduced in Android 4.3.

The **KeyChain API** allows applications to have access to the system-wide credential storage in order to use or protect private keys and certificate chains. This enables several applications to share the same credentials with the consent of the user.

The **Android Keystore provider** is a custom Java Security Provider that allows applications to generate and use their own credentials that cannot be accessed by other applications. This new security provider can be used with three JCA (Java Cryptography Architecture) classes: KeyStore, KeyPairGenerator and KeyGenerator. The use of standard APIs avoids to create yet another security-specific API, and therefore provides an easy way for developers to integrate security into their applications.

Keystore service

The Keystore service was originally implemented by a single native daemon called *keystore*. In version 4.3, the *keystore* daemon was replaced by a centralized service based on the Binder architecture. In such architecture, IPC between the 'application framework' (`KeyStore.java`) and the 'system service' (`Keystore.cpp`) is achieved via the binder proxy `IKeyStoreService`.

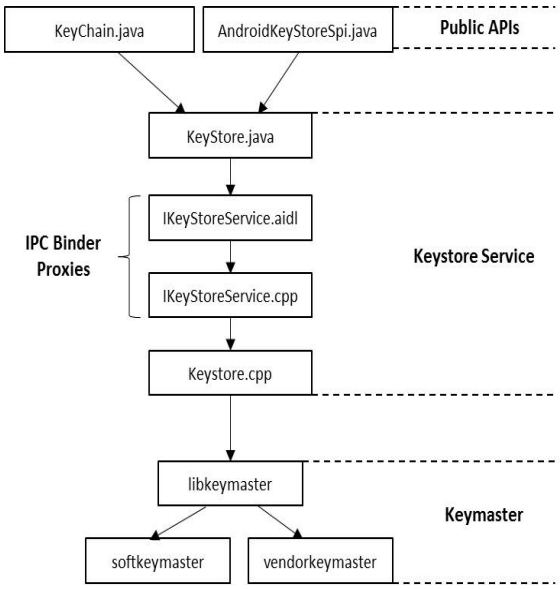


Figure 2.3: Android KeyStore Architecture

Started at device boot, the keystore service registers itself to `ServiceManager` with the name `android.security.keystore`. Aside from key generation and signing/verifying data, it provides all other KeyStore-related operations. It provides two main functionalities: keys storage and mitigation of keys misuse. For **secure key storage**, we discuss this more detail in chapter 4.

Concerning **key use authorization**, the Keystore service asks applications to specify authorized uses of their keys. Authorizations are then enforced whenever the key is used. The enforced authorizations are divided into three categories: (1) *cryptography*: authorized operations (encrypt, sign); (2) *temporal validity*: expiration time; and (3) *user consent*: explicit or implicit user authentication.

Keymaster

The keymaster, introduced in Android 4.1, is a HAL module that is dynamically loaded by the Keystore service. The keymaster accomplishes two tasks.

First, it simplifies the integration of hardware-backed implementations. The HAL allows vendors to provide their own proprietary solutions without modifying or affecting the higher level system. The only requirement for vendor-specific solutions to satisfy is to implement the HAL interface defined in `keymaster.h` and including the following seven operations:

1. `generate_keypair`,
2. `import_keypair`,
3. `sign_data`,
4. `verify_data`,
5. `get_keypair_public`,
6. `delete_keypair`,
7. `delete_all`.

Android 6.0 contains two HAL interfaces: `keymaster0.h` and `keymaster1.h`. The newly defined `keymaster1.h` has been significantly enhanced with new cryptographic primitives (AES and HMAC) and access control. At the writing of this chapter, the Keystore service of the build `android-6.0.0_r22` still uses the `keymaster0.h`.

Second, it protects keys material from extraction. The keymaster could be seen as a device to which the Keystore service delegates all security-sensitive operations. Indeed, the keys material is never exposed outside the keymaster device, and therefore, the application process never manipulates the keys directly. Thus, an adversary who may compromise an application will not be able

to read its keys. Hardware-backed keymasters provide better security: they protect keys from extraction even if the Android OS is compromised. We say that keys are **device-bound** because an adversary compromising the OS still cannot export them from the user's mobile, although she may be able to use them.

2.4 Secure Hardware Technologies in Smartphones

Now, we review the hardware technologies used to build trusted execution environments inside smartphones. In this thesis, we only consider the most standard ones which, in order of decreasing presence inside devices, are: the Baseband, the Secure Element and ARM TrustZone.

2.4.1 The Baseband

Modern smartphones consist of two separate, but cooperating, systems: the application processor and the baseband processor. This common design is found across almost all chipset manufacturers (e.g. [Qua16b, Sam16]). The application processor usually comes in form of a System on a Chip (SoC) design. It runs the smartphone operating system, such as Android, and contains the peripherals such as the display, touchscreen and storage. The Baseband (cellular modem) processor is the communication component to the cellular network.

The Baseband is mostly composed from an ARM central processing unit CPU, a digital signal processor (DSP) and the necessary radio components such as the signal amplifier. The type of baseband processor depends highly on the actual device manufacturer and the kind of cellular network that the device is built for (e.g. GSM, LTE, etc.). The baseband processor runs a specialized real-time operating system (RTOS) with no isolation mechanisms or memory virtualization. This means that all the running tasks inside the baseband are present in the same address space. Multiple applications are executed within the baseband RTOS: management of radio interface and implementation of communication interfaces between the application processor and various hardware extensions (e.g., SIM cards). Almost all baseband architectures are based on ARM architecture.

The application processor and the baseband processor are connected to each other on the device main board. In order to reduce costs, chipset manufacturers sometimes integrate both into one single chip, but they still function independently. This allows for better flexibility for various smartphone manufacturers. Within the standard TS 27.007 [ETS11], 3GPP [3GP16] defines the command set, called Hayes Attention (AT) commands, used to interact with the baseband. Briefly, AT commands behave behave as an asynchronous messaging interface. The structure of AT commands is simple: they consist of standard text strings containing a prefix, the command symbol and the related parameters. For instance, 3GPP defines these three AT commands to communicate with SIM card applications:

1. `AT+CCHO` to open a logical communication channel with the a specific application inside the SIM card,
2. `AT+CGLA` to send commands using the opened logical channel,
3. `AT+CCHO` to close the logical channel.

Considered as a critical resources, the design of basebands involves stringent security requirements. Indeed, they have to be certified by multiple institutions before they are allowed to operate on public cellular networks [Com16a]. Because the process of development and certification is very costly, there are only a few baseband manufacturers [Kun16]. The primary security mechanism of the baseband RTOS is the strong isolation from the main rich operating system. Indeed, as said above, they both run on a separate memory and have their own resources. Thus, we might

believe that the RTOS cannot be affected by malicious malware executing inside the main operating system. However, this assumes that the communication interface has been well designed and cannot be exploited to breach the RTOS security. Unfortunately, despite its significant role in smartphone security, a few research work has focused on the security issue of the baseband. The absence of such work most likely roots from the closed nature of this technology where manufacturers regard every detail as a secret. Recent studies, like [MGS11] and [Wei12], have shown that the communication interface of the baseband contains some security vulnerabilities that might lead to compromise the whole smartphone. For instance, authors in [Wei12] have demonstrated some memory corruption flaws inside the baseband stack that can be remotely exploited to inject an arbitrary code on the baseband processor. This shows that security by strong hardware isolation is not enough to guarantee a trustworthy execution environment.

2.4.2 The Secure Element

A Secure Element (SE) is a tamper-resistant smart card that is integrated inside smartphones. It is either embedded in the device or inserted into a dedicated or an SD card slot. An SE is essentially a minimal computation environment with volatile and non-volatile memories. It also includes cryptographic coprocessors that improve the execution time of some common encryption algorithms [RE10]. Secure Elements use various techniques to implement tamper resistance, making it quite hard to extract data by disassembling or analyzing the chip. In addition, they come with multi-application operating system that ensures that strong isolation between applications [Mar98].

The design goal of Secure Elements is to bring the advantage of smart cards to smartphones. Indeed, nowadays, smart cards are being used in various applications ranging from ticketing for public transportation to credit cards and VPN (Virtual Private Network) credential storage. Since SEs are actually smart cards, they can potentially be used for any application intended for physical cards. Moreover, due to their multi-application nature, only one single SE can potentially host all the applications that people use daily.

Secure Elements offer two ways to enhance the security of applications dealing with secret data or executing sensitive algorithms. The first way is to host the entire application. Only quite simple applications can benefit from such design because of the limited resources of SEs. This includes One Time Password (OTP) generators and payment applications. The second way is to logically split the security-critical part of an application from the rest, and to implement that specific part inside the SE. This concerns complex applications requiring to interact with users via graphical interface. In such model, an SE application is composed of two parts interacting with each other:

- *off-card application*: this part resides on the smartphone and it usually implements the user interface or any resource-demanding algorithm.
- *on-card application*: this part is installed on the SE and it protects the sensitive information (i.e. data or algorithm) of the whole application.

As a matter of fact, the driving force behind SE deployment was to implement services that could be used in a contactless way. This allows users, for instance, to have their mobile devices act as debit/credit cards. To this end, SEs generally come coupled with the Near Field Communication (NFC) technology that enables smartphones to emulate contactless smart cards [COO11]. In this mode, the devices receive commands over NFC, then the NFC component forwards them to the integrated SE that processes them and sends replies again over NFC. Despite its great interest, the Android security overview [And16c] explicitly states that 'low level access to the SE is not available to third-party apps' in order to avoid remote denial of service attacks. Such security measurement does not make SEs easily accessible via other interfaces than NCF, which hinders their adoption for different applications like VPN credential storage.

Secure Elements come in different flavors in smartphones. The most widely deployed form of SE is the Universal Integrated Circuit Card (UICC, commonly known as *SIM card*). Before being an SE, SIM cards were originally invented for user authentication to cellular network. This is why they are only connected to the baseband processor, and not to the application processor that runs the main operating system (e.g. Android). All communications go through a proprietary IPC interface to the baseband. In addition, they are carried out using AT commands which are actually supported by the Android Telephony Manager or the SEEK for Android project [Gie16] implementing the SIMalliance Open Mobile API specifications [SIM15].

Below, we review some technical details about programming in SE: format of commands and JavaCard.

Communication Model

Similarly to smart cards, Secure Elements never send data without an external stimulus. They only respond to commands received from another program (e.g. card reader). In other words, communication is always initiated by the client system. This yields a pure master-slave relationship with the SE as slave.

Secure Elements receive data as a record called APDU (Application Protocol Data Unit). that contains the description of the invoked command and its arguments. The SE replies to the command APDU (C-APDU) by another type of record referred to as the Response APDU (R-APDU). The structure of APDU is defined by ISO/IEC 7816-4 [ISO13]. As shown in Figure 2.4, the C-APDU consists of a header and a body. The header includes the following fields:

- `CLA` and `INS` specifying the application class and instruction,
- `P1` and `P2` qualifying specific instructions.

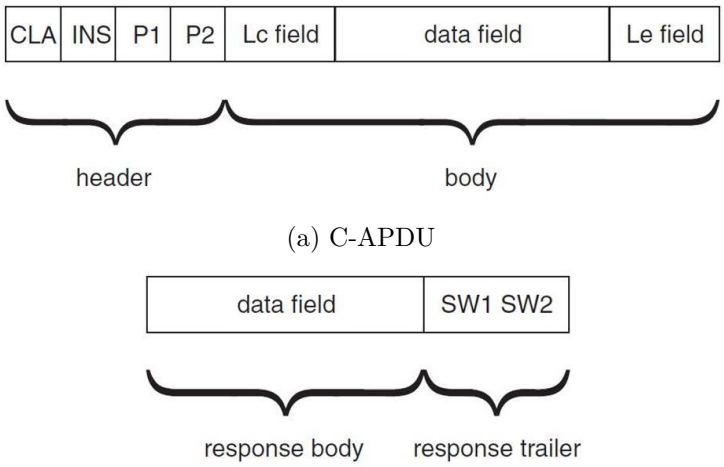
The body of the APDU is used to transmit data to the SE. Since it can vary in size, the `Lc` field specifies the size of the data field (in bytes). The `Le` field specifies the number of bytes expected to be returned by the SE.

Regarding the R-APDU, it has a much simpler structure. It consists of a body and a trailer. The body is either empty or includes a data field the length of which is determined by the `Le` field in the corresponding C-APDU. The trailer consists of two bytes of status information called `SW1` and `SW2`. In general, `SW1` represents the error category and `SW2` represents a command-specific error indication.

JavaCard

JavaCard constitutes one of the greatest successful stories in the history of computer science. In 2014, more than 12 billion of JavaCard have shipped to date [Jav14]. JavaCard [Che00] is Java-based SE. It is designed by naively adding a lightweight Java bytecode interpreter to the SE system and downloading Java class files which were previously converted to a specific format beforehand. JavaCard brings the main advantages of Java to SE software development: object-oriented programming, strongly typed language, interoperability (Write once, run everywhere), etc. In addition, JavaCard provides a good basis for programmable multi-application Secure Elements, since it can dynamically load more than one application on the SE system.

JavaCard technology was tailored in order to enable Java bytecode to run on SEs. Due to resources constraints, two design choices were made. Firstly, the Java Virtual Machine (JVM) is split into two parts: the converter running outside the SE and the bytecode interpreter running inside the SE. Thus, in JavaCard, many tasks are performed outside the trusted SE, such as classes loading, links resolution and optimization. Secondly, every component of the original Java platform was significantly reduced. Indeed, most deployed JavaCard provide support for



(a) C-APDU

(b) R-APDU

Figure 2.4: Overview of the APDU Structure

primitive data types such as `boolean`, `byte` and `short`, but not large data such as `long` and `double`. Some JavaCard support `int`. JavaCard supports one-dimensional arrays, packages, interfaces and exceptions. The supported subset of the Object Oriented (OO) paradigm includes inheritance, virtual methods, dynamic object creation, etc. However, the unsupported features include dynamic class loading, garbage collection and object serialization.

The system architecture of the Java Card is mainly composed of three layers: the JavaCard Virtual Machine (JCVM) including the bytecode interpreter, the JavaCard Framework providing a well-structured framework to access the system-level services, and finally the JavaCard firewall which isolates the different applets present on the card from each other. The JavaCard specification does not specify the mechanism of installing, updating, or deleting applications. The application management is defined by GlobalPlatform card specification.

2.4.3 ARM TrustZone

ARM Architecture

ARM architecture is the most dominant architecture used in smartphones [ARM16a]. It has evolved over time through several versions (e.g., ARMv1, ARMv2 ... ARMv8). Each architecture version includes families or profiles (e.g., ARM11 or Cortex-A). ARM processors have two classes of software execution: (1) *privileged* in which the running software can use all the instructions and has access to all hardware resources, and (2) *unprivileged* in which software has restricted access to instructions and some resources such as the memory and the peripherals. Within these two classes, ARM supports different modes of execution. For instance, ARMv8 supports nine modes: user, system, supervisor, monitor, Hyp, interrupt request, fast interrupt, abort, and undefined. The mode of execution determines the privilege level of the executed code. As depicted in Figure 2.5, four privilege levels are available in ARMv8 AArch64:

1. `EL0` for user applications,
2. `EL1` for operating system kernels,
3. `EL2` for hypervisors,
4. `EL3` for TrustZone monitors.

In ARM architecture, an exception is an interruption of the current execution. It involves the suspension of the normal execution, the change of the execution mode and the transfer of control to a handler in a predefined entry point. The first operation done by a handler is to save the

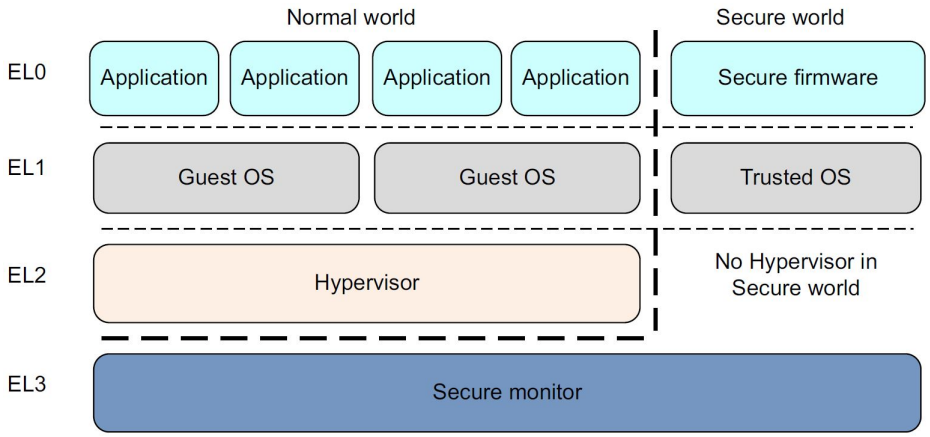


Figure 2.5: ARMv8 Execution Levels [ARM15a]

registers which are shared between the different modes, so the interrupted application can resume at the end of the handler execution. ARM has two types of interrupt known as `IRQ` (Interrupt ReQuest) and `FIQ` (Fast Interrupt reQuest). As their name indicates, `FIQ` interrupts have higher priority and more banked registers than `IRQ` interrupts.

ARM processors use coprocessors to extend their architecture. They allow up to sixteen coprocessors. Coprocessor 15 (`CP15`) is reserved for special functions such as memory management, system performance monitoring and TrustZone configuration.

TrustZone

ARM TrustZone is a set of hardware extensions present in ARM high-end processors such as the Cortex series [ARM09]. Integrated into the CPU core, TrustZone enables trusted computing without the need of extra hardware chips. It divides the platform into two worlds: a secure world for the execution of trusted operating system and a normal world for the execution of traditional operating system. A CPU core that is TrustZone-compliant can be seen as two virtual CPU cores with different privileges and a strictly controlled communication interface. The secure world is a privileged state of the processor. It gives access to additional control registers. The Secure Configuration Register (`SCR`) is one of these protected registers and it exists in the `CP15` coprocessor. It allows the secure world to control the behavior of the normal world. Indeed, it can intercept all the interrupts, including those of the normal world.

TrustZone is orthogonal to the ARM privilege levels (i.e. `EL0`, `EL1`, ..., etc.). Thus, each world includes a kernel and a user space. One special mode is the monitor mode. It is shared between the two worlds and it enables the processor to switch between the two worlds. Therefore, the monitor mode is used to communicate between the two worlds. It is responsible for saving and restoring state during a world transition.

Below, we discuss the different components that constitute the architecture of TrustZone. Figure 2.6 shows an overview of TrustZone architecture.

Core Enhancements. TrustZone enables a secure software environment by introducing several modifications to the core architecture. These modifications can be summarized as follows:

- *NS bit* of a newly introduced register in `CP15` coprocessor. This bit determines the state of the processor (i.e. its executing world) and it is only accessible in the secure world;
- *monitor mode* which is a new privileged mode. The monitor mode is powerful because it has a full view of all resources, both secure and non-secure. It is used to switch between

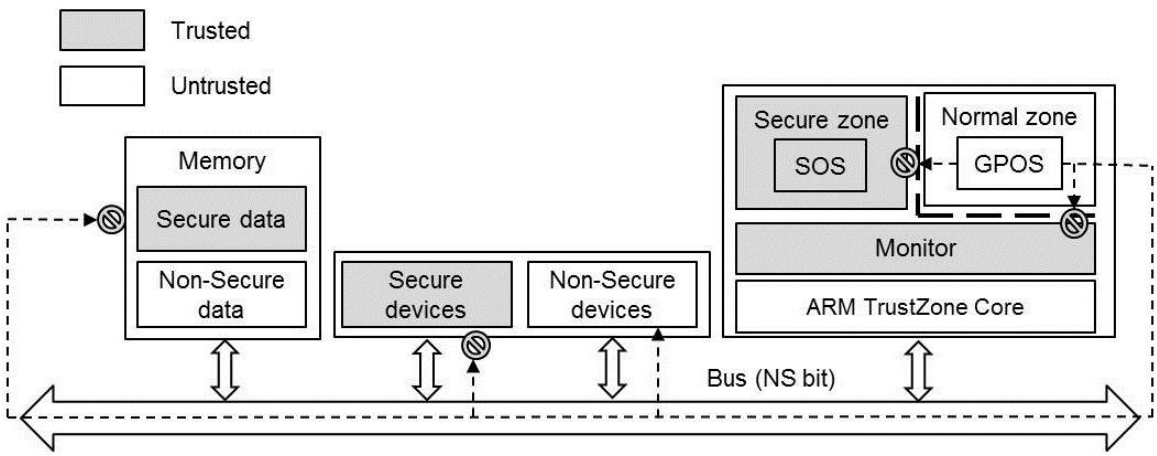


Figure 2.6: An Overview of the ARM TrustZone Technology

the secure and the normal world. The monitor mode can be reached only in two ways: by means of a new privileged instruction, the Software Monitor Call (`SMC`), and by configuring exceptions to be handled in monitor mode;

- The NS bit is propagated through the system bus indicating the state of the processor. This allows to partition the memory into the secure and the normal world. Likewise, the peripherals can be configured to be accessible only by the secure world by configuring their related DMA (Direct Memory Access);
- Since each world must be able to handle its own exceptions, the base of the exception vector table can be assigned independently for the secure and normal worlds. In addition, a separate table is used for the monitor mode.

Additional Components. Besides the enhancements to the ARM core, TrustZone includes some additional components, namely:

- *TrustZone protection controller* to define the security of memory regions and memory mapped devices;
- *TrustZone interrupt controller* to define which interrupt instruction must be treated as secure interrupts;
- A family of bridges and memory adapters connected to the protection controller in order to check the validity of every access request against the security attributes that were previously specified in the corresponding protection controllers.

2.5 Conclusion

In this chapter, we described the core concepts of smartphone security. We also discussed the difficulties challenging the security challenges of such devices. Then, we presented three three technologies that help overcome the related difficulties, and thus provide better security.

In this thesis, we aim to build trusted solutions for smartphones. This involves not only leveraging hardware primitives due to their strong security guarantee, but also some trusted models especially tailored to protect sensitive applications in smartphones. In chapter 6, we define the underlying trusted model that we will keep using in chapters 7 and 8. In these chapters, being

more advantageous than other technologies, we particularly focus on one hardware primitive: ARM TrustZone. Chapter 7 precisely defines how ARM TrustZone can be combined together with our model of chapter 6 to build a trusted execution environment from smartphones. We show in chapter 8 that our construction is powerful and can be integrated into advanced architectures. Of particular interest, we design a trusted execution environment leveraging ARM TrustZone that can protect its sensitive data even when its secure kernel is compromised.

Chapter 3

Cryptographic Primitives and Provable Security

"Let's start at the very beginning
A very good place to start
When you read you begin with A-B-C
When you sing you begin with do-re-mi"

– The Sound of Music, Do-Re-Mi

Contents

3.1 Cryptographic Primitives **32**
3.1.1 Notations 32
3.1.2 Block Ciphers 32
3.1.3 Tweakable Block Ciphers 33
3.1.4 Encryption Schemes and Mode of Operations 33
3.1.5 Nonce-Based Symmetric Encryption 36
3.1.6 Encoding Schemes 37
3.1.7 Message Authentication Codes 37
3.1.8 Authenticated Encryption 38
3.2 Provable Security **39**
3.2.1 Overview of Provable Security 40
3.2.2 Functions and Permutations 42
3.2.3 Threat Models for Symmetric Encryption 43
3.2.4 Confidentiality of Symmetric Encryption 44
3.2.5 Integrity of Symmetric Encryption 45
3.2.6 Indistinguishability from Random Bits 46
3.2.7 Strong Unforgeability of MAC 47
3.2.8 Results for Generic Compositions 47

This chapter introduces the required theoretical background necessary for the comprehension of this thesis. Firstly, we introduce the cryptographic primitives that we will use in later chapters. In the second part of the chapter, we introduce provable security which is of paramount importance for modern cryptography, and then we provide definitions of security models for symmetric encryption. Readers who are familiar with these topics may skip this chapter.

3.1 Cryptographic Primitives

Cryptographic primitives are algorithms designed to perform some specific cryptographic operations. A primitive is used as a building block to create much larger and more complex cryptographic protocols which aim to achieve a variety of security goals. Generally, two main groups of primitives are considered: asymmetric (commonly referred to as public-key) and symmetric. In this thesis, we will focus solely on symmetric cryptography. We begin by providing the formal definitions of some basic symmetric primitives.

3.1.1 Notations

First and foremost, we provide the notation symbols that we will use throughout this thesis.

A message is a string. A string is an element of $\{0,1\}^*$. The concatenation of strings X and Y is denoted $X||Y$ or simply XY. For a string X, its length is represented by $|X|$. For an integer $N \in \mathbb{N}$, N++ denotes the C-like ++ operator that returns the value N and then increases its value by 1.

If f is a probabilistic (resp., deterministic) function, then $y \xleftarrow{R} f(x)$ (resp., $y \leftarrow f(x)$) denotes the process of running f on input x and assigning the result to y. Given an algorithm $\mathcal{A}$, the notation $\mathcal{A} = x$ means that the algorithm $\mathcal{A}$ outputs the value x.

3.1.2 Block Ciphers

Block ciphers are one of the most important and widely used primitives in modern cryptography. The primary goal of block ciphers is to provide data confidentiality. Nevertheless, it is also used to help build other primitives, such as message authentication code (MAC).

The concept of block ciphers is derived from the notion of *permutation*. A permutation is a one-to-one function, hence invertible, from one set to itself. A block cipher is a function $E : \mathsf{Key} \times \{0,1\}^l \longrightarrow \{0,1\}^l$, where Key is a finite nonempty set and for a fixed k, $E(\mathsf{k},.)$ is a permutation on $\{0,1\}^l$. Hereafter, we denote $E(\mathsf{k},.)$ as $E_{\mathsf{k}}(.)$. We notice that a block cipher works on fixed length inputs known as *blocks*. Hence, the number l is called the *block size*. More formally, a block cipher can be defined as being a family of permutations on $\{0,1\}^l$ indexed by keys $\mathsf{k} \in \{0,1\}^k$. We define a block cipher E to be a set $\{E_{\mathsf{k}} : \mathsf{k} \in \mathcal{K}(k)\}$, where $\mathcal{K}(k)$ is a finite nonempty keyspace depending on k which is known as the security parameter. For the sake of simplicity, k may be regarded as the key length. Being a permutation, for every $E_{\mathsf{k}}(.) \in E$ there exists an inverse permutation which we denote $E_{\mathsf{k}}^{-1}(.)$. Conventionally, $E_{\mathsf{k}}(.)$ denotes the encryption operation of the block cipher, while $E_{\mathsf{k}}^{-1}(.)$ denotes the decryption one. We require that for all $\mathsf{k} \in \mathcal{K}(k)$, E_{k} and E_{k}^{-1} are efficiently computable. We will examine more closely how to model the security of a block cipher in section 3.2.2.

The three most widely known block ciphers are the (1) Data Encryption Standard (DES), (2) triple DES (3DES) and the (3) Advanced Encryption Standard (AES). DES was standardized in 1976 by the American National Bureau of Standards[1] [oS77]. It is a 64-bit block cipher with 56-bit keys following a Feistel cipher structure [Fei73]. Due to the relatively short key size, DES is presently vulnerable to brute-force attacks. In 1998, the Electronic Frontier Foundation released details of its "DES Cracker", which is a dedicated machine capable of cracking DES in less than three days. In order to withstand brute-force attacks, Triple DES (3DES) was proposed. It uses the DES algorithm three times in an encrypt-decrypt-encrypt pattern with either two or three independent keys. By using multiple keys, the effective key size of the scheme is increased. 3DES was eventually included as part of the Data Encryption Standard in 1999.

[1] now named NIST (National Institute of Standards and Technology)

The National Institute of Standards and Technology (NIST) announced a competition to find the successor for DES in 1997. The proposed alternative was required to have the same level of security as 3DES, but be more efficient. The winning block cipher, to be known as AES (the Advanced Encryption Standard) [oSN01], was submitted by Vincent Rijmen and Joan Daemen. Instead of the Feistel structure of DES, AES uses a substitution-permutation network. In addition, it has a block size of 128 bits and works with keys of length 128, 192 or 256 bits.

3.1.3 Tweakable Block Ciphers

A tweakable block cipher (TBC) is a block cipher that admits an additional public input that is called the "tweak". The tweak serves to introduce extra variability at the message-block level. Tweakable block ciphers have been formalized by Liskov, Rivest and Wagner [LRW02].

The motivation for such a construction is that, in many situations, independent instances of a block cipher are desirable. Indeed, block ciphers are deterministic and applying the block cipher on identical message blocks gives identical ciphertext blocks. However, if an independent instance of the block cipher were to be used for each block, then this would not be the case. One potential solution to this particular issue would be to re-key the block cipher with fresh, random and independent keys for each block processing. This is hard to realize in real-world systems. In addition, most block ciphers incur a cost associated with changing the key, so a major motivation of Liskov et al. was to accomplish independent instances of a block cipher in a more efficient way.

Regarding syntax, a tweakable block cipher is a function $\widetilde{E} : \mathsf{Key} \times \mathsf{Tweak} \times \{0,1\}^l \longrightarrow \{0,1\}^l$, where, as their names indicate, Key is the key space and Tweak is the tweak space. We require that for every $\mathsf{K} \in \mathsf{Key}$ and $T \in \mathsf{Key}$, $\widetilde{E}(\mathsf{K}, T, .)$ is a permutation, hence invertible, on $\{0,1\}^l$. The value l is called the block size. Henceforth, we denote $\widetilde{E}(\mathsf{K}, T, .)$ by $\widetilde{E}_{\mathsf{k}}(T, .)$, and the inverse of this permutation by $\widetilde{E}_{\mathsf{k}}^{-1}(T, .)$. More formally, a tweakable block cipher $\widetilde{E}$ is a family of permutations indexed by a pair $(\mathsf{K}, T) \in \mathsf{Key} \times \mathsf{Tweak}$. This means that for a fixed key K, the two permutations $\widetilde{E}_{\mathsf{k}}(T1, .)$ and $\widetilde{E}_{\mathsf{k}}(T2, .)$ are independent for all $T1, T2 \in \mathsf{Tweak}$ when $T1 \neq T2$. There is, however, a semantic asymmetry between the key and the tweak: the key is secret and gives rise to security, while the tweak may be public and gives rise to variability.

There exist multiple ways to build a tweakable block cipher. Mercy [Cro01] is an early example of block ciphers supporting tweaks in a native manner. It is also possible to build tweakble block ciphers from conventional block ciphers, as in Rogaway's XE and XEX modes [Rog04a], or by modifying existing block cipher designs [GHL$^+$07]. Similarly to block ciphers, TBC can be used not only to construct encryption schemes, but also to build hash functions and message authentication codes.

3.1.4 Encryption Schemes and Mode of Operations

A symmetric encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms: $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{D}$. The *key generation* algorithm $\mathcal{K}$ is probabilistic and takes as input a security parameter k to output a key K. We write $\mathsf{K} \xleftarrow{R} \mathcal{K}(k)$. The *encryption algorithm* $\mathcal{E}$ may be randomized. It takes as input a key K and a plaintext message $m \in \mathcal{M}$, where $\mathcal{M}$ is the message space, and returns a ciphertext $c \in \mathcal{C}$, where $\mathcal{C}$ is the ciphertext space. We suppose that both $\mathcal{M}$ and $\mathcal{C}$ are finite nonempty sets of strings. We write $c \longleftarrow \mathcal{E}_{\mathsf{k}}(m)$. The *decryption algorithm* $\mathcal{D}$ is deterministic. It takes as input a key K and a ciphertext $c \in \mathcal{C}$ to return either a plaintext $m \in \mathcal{M}$ or a special symbol $\perp$ to indicate that the ciphertext is invalid. We require that $\mathcal{D}_{\mathsf{k}}(\mathcal{E}_{\mathsf{k}})(m) = m$ for all m and K.

As mentioned previously, symmetric encryption schemes are based on block ciphers. However, a block cipher only encrypts or decrypts blocks of data of fixed length. Therefore, we use blocks in modes of operation to build encryption schemes which can perform on multiple blocks of data. The most obvious way to encrypt multiple blocks would be to split the data into a sequence of blocks and then separately encrypt each one of them using the block cipher. This particular mode

of operation is known as Electronic Code Book (ECB) mode. This mode is not used in practice as it is not secure. We will see later in section 3.2 that ECB does not satisfy the property of *semantic security*. Informally speaking, since blocks ciphers are deterministic, the encryption of two identical plaintext blocks will result in two identical ciphertext blocks. An eavesdropper, who is an attacker intercepting the exchanged ciphertexts, might use this relation in order to deduce the structure of the corresponding plaintext. In order to avoid this problem, other modes were proposed. The ISO/IEC standard 10116 [ISO06] defines five modes of operation: Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output FeedBack (OFB) and Counter (CTR) mode. In this thesis, we focus solely on two of these modes of operation: CBC and CTR modes.

CBC Mode

CBC mode is perhaps the most widely deployed mode of operation. In particular, it has the advantage of being well adapted to constrained cryptographic devices, such as smart cards. Indeed, it can perform encryption and decryption as soon as blocks of data are received, which is quite useful for devices with quite small buffers. In addition, small devices do not have enough computation power to support multi-threading, and therefore they are not impacted by the fact that CBC mode is not parallelizable (i.e. encryption cannot be split into parts that run in parallel).

As with most modes of operation, it is necessary to split a plaintext m into a sequence of blocks $m_1 \,||\, m_2 \,||\, ... \,||\, m_n$. Without loss of generality, we assume that the length of our plaintext m is a multiple of the block size, otherwise some padding is required. *Padding* refers to some mechanisms that complete the plaintext with some data, so that its length is a multiple of the block size. Afterward, encryption is done by XORing the plaintext m_i with the previous ciphertext block c_{i-1} before applying the block cipher E_{k}. Decryption is performed by simply reversing this operation: the block cipher E_{k}^{-1} is first applied on the ciphertext block c_i, and then the output is XORed with the previous ciphertext block c_{i-1} in order to obtain the plaintext block m_i.

As described above, we note that the encryption and the decryption operations require a special block ciphertext c_0. This block is called the initialization vector (IV). Each ciphertext must have an associated IV which should be known by both the sender (encryptor) and the receiver (decryptor). There are many ways to choose and share the IV. The most basic one is to include it as an extra parameter to the encryption and decryption algorithms. A formal definition is given below.

Definition 3.1. [*The IV Cipher Block Chaining (IV CBC) Encryption Scheme*]
Let E be a block cipher of block size l. Let ivCBC$[E] = (\mathcal{K}\text{-ivCBC}, \mathcal{E}\text{-ivCBC}, \mathcal{D}\text{-ivCBC})$ be its associated IV CBC encryption scheme. Given an initialization vector $iv \in \{0,1\}^l$ and a message $m \in \{0,1\}^{ln}$, where $n \in \mathbb{N}$, the encryption and the decryption algorithms are defined as follows:

IV CBC Encryption $\mathcal{E}_{\mathsf{k}}$-ivCBC(iv, m)

1: Parse m as $m_1||...||m_n$, where $|m_i| = l$
2: $c_0 \longleftarrow iv$
3: **for** $i = 1...n$ **do**
4: $\quad c_i \longleftarrow E_{\mathsf{k}}(c_{i-1} \oplus m_i)$
5: **end for**
6: **return** $c_1 \,||\, ... \,||\, c_n$

IV CBC Decryption $\mathcal{D}_{\mathsf{k}}$-ivCBC(iv, c)

1: Parse c as $c_1||...||c_n$, where $|c_i| = l$
2: $c_0 \longleftarrow iv$
3: **for** $i = 1...n$ **do**
4: $\quad m_i \longleftarrow E_{\mathsf{k}}^{-1}(c_i) \oplus c_{i-1}$
5: **end for**
6: **return** $m_1 \,||\, ... \,||\, m_n$

One point should be underlined in the definition. We make the simplifying assumption that $\mathcal{D}_{\mathsf{k}}$-ivCBC$(.,.)$ never returns the error message $\perp$. Indeed, it takes any ciphertext as input, and always returns some string.

Now, let's define the *stateless* CBC which is the most popular variant of the CBC mode based on the IV CBC one. Other variants will be defined in section 3.1.5. The *stateless CBC* consists of choosing a fresh, random IV for every encrypted message and then sending it as the first block of ciphertext. Stateless CBC was proven to be secure by Bellare et al. [BDJR97] in standard and commonly accepted security models. We will discuss these models in section 3.2.

Definition 3.2. [*The Stateless Cipher Block Chaining (CBC) Encryption Scheme*]
Let E be a block cipher of block size l. Let ivCBC$[E] = (\mathcal{K}\text{-ivCBC}, \mathcal{E}\text{-ivCBC}, \mathcal{D}\text{-ivCBC})$ be its associated IV CBC encryption scheme. Given a message $m \in \{0,1\}^{ln}$, where $n \in \mathbb{N}$, we define the stateless CBC scheme CBC$[E] = (\mathcal{K}\text{-CBC}, \mathcal{E}\text{-CBC}, \mathcal{D}\text{-CBC})$ as follows:

CBC Encryption $\mathcal{E}_{\mathsf{k}}$-CBC(m)

1: $c_0 \xleftarrow{R} \{0,1\}^l$
2: $c \longleftarrow \mathcal{E}_{\mathsf{k}}\text{-ivCBC}(c_0, m)$
3: **return** $c_0 \,||\, c$

CBC Decryption $\mathcal{D}_{\mathsf{k}}$-CBC(c)

1: Parse c as $c_0 \,||\, c_1 \,||\, ... \,||\, c_n$, where $|c_i| = l$
2: $m \longleftarrow \mathcal{D}_{\mathsf{k}}\text{-ivCBC}(c_0, c_1 \,||\, ... \,||\, c_n)$
3: **return** m

It is worth mentioning that the random IV is denoted $\mathbf{c_0}$ in order to highlight the fact that the IV is included along with the ciphertext.

CTR Mode

The other mode of operation that we study in this thesis is the Counter (CTR) mode. Encryption with CTR was firstly proposed by Diffie and Hellman in [DH79]. This mode is widely implemented due to its significant efficiency advantages over the other modes without weakening the security. Indeed, the mode works by encrypting a counter value using the block cipher and XORing the output with the plaintext block. Therefore, unlike CBC mode, CTR mode is parallelizable. There exist different variants of CTR-mode encryption; we are only concerned by the *blockwise randomized counter mode* here.

Notation. Let $F : \mathtt{Key} \times \{0,1\}^l \longrightarrow \{0,1\}^l$ be a block cipher, where $\mathtt{Key}$ is a finite nonempty set, and $F_{\mathsf{k}}(.) = F(\mathsf{k}, .)$ is a pseudorandom function mapping l-bit strings to l-bit strings for a fixed $\mathcal{K} \in \mathsf{Key}$. Given a message $X \in \{0,1\}^l$ and an unsigned positive integer i, $X + i$ denotes the l-bit message which is obtained from adding i modulo $2^{l/2}$ to X treated as a number (msb[2] first, lsb[3] last), taking the result modulo 2^l and then returning it back into l-bit message.

Operation. To encrypt a message $m \in \{0,1\}^{ln}$, the CTR mode can be viewed as a way of generating a pseudorandom stream from a block cipher. First, a random IV is chosen; hereafter, this IV is denoted $\mathtt{ctr}$. Then, a pad is generated by computing $r_i \leftarrow F_{\mathsf{k}}(\mathtt{ctr} + i)$. Finally, the ciphertext is computed as $c \leftarrow \mathtt{ctr}||c_1||...||c_n$, where $c_i \longleftarrow m_i \oplus r_i$ for $i = 1$ to n. The decryption operation is computed using the same relations, and thus it does not require F_{k} to be invertible.

Definition 3.3. [*The CTR Encryption Scheme*]
Let F_{k} be a pseudorandom function on $\{0,1\}^l$. Let CTR$[F] = (\mathcal{K}\text{-CTR}, \mathcal{E}\text{-CTR}, \mathcal{D}\text{-CTR})$ be its associated CTR encryption scheme. Given a message $m = m_1||...||m_n \in \{0,1\}^{ln}$, the encryption and the decryption algorithms are defined as follows:

[2]Most Significant Bit
[3]Least Significant Bit

CTR Encryption $\mathcal{E}_{\mathsf{k}}$-CTR(m)

1: Parse m as $m_1||...||m_n$, where $m_i = l$
2: $\mathtt{ctr} \xleftarrow{R} \{0,1\}^{l/2}$
3: $c_0 \longleftarrow \mathtt{ctr}\,||\,0^{l/2}$
4: **for** $i = 1...n$ **do**
5: $\quad c_i \longleftarrow m_i \oplus F_{\mathsf{k}}(c_0 + i)$
6: **end for**
7: **return** $c_0||c_1||...||c_n$

CTR Decryption $\mathcal{D}_{\mathsf{k}}$-CTR(c)

1: Parse c as $c_0||c_1||...||c_n$, where $c_i = l$
2: **for** $i = 1...n$ **do**
3: $\quad m_i \longleftarrow c_i \oplus F_{\mathsf{k}}(c_0 + i)$
4: **end for**
5: **return** $m_1||...||m_n$

We emphasize two points in this definition. First, the random `ctr` is denoted $\mathbf{c}_0$ in order to underline the fact that `ctr` is sent as part of the ciphertext. Second, the variant described here operates only on messages of at most $2^{l/2}$ blocks (assuming that l is even). Otherwise, two blocks of the ciphertext would be encrypted by the same pad, which compromises the security of this mode. More precisely, if the counter value wraps around, then two blocks of ciphertext will be encrypted with the same counter value. This means that two blocks of plaintext are XORed with the same bit string (i.e. $F_{\mathsf{k}}(\mathtt{ctr})$) due to the deterministic property of the block cipher. Using this observation, some attacks become possible. For instance, consider the ciphertexts $c_1 = m_1 \oplus F_{\mathsf{k}}(\mathtt{ctr})$ and $c_2 = m_2 \oplus F_{\mathsf{k}}(\mathtt{ctr})$ which have been encrypted with the same `ctr`. We can easily see that $c_1 \oplus c_2 = m_1 \oplus m_2$. Hence, the security is compromised, since the XOR result of two plaintexts, m_1 and m_2, can be used to deduce some information about these two plaintexts.

3.1.5 Nonce-Based Symmetric Encryption

Rogaway in [Rog04b] introduces the concept of nonce-based symmetric encryption. It is about a variation in the syntax of symmetric schemes in which encryption and decryption take a *nonce* as an extra input. A nonce is a non-repeating value. Rogaway motivates his work by arguing that nonce-based schemes are more practice-oriented. Indeed, the nonce is not needed to be neither random, nor secret or unpredictable. In fact, it can be even generated by the user as long as no value is ever repeated. Thus, nonce-based schemes alleviate the security requirements to handle auxiliary inputs, such as the random IV in the stateless CBC, and therefore nonce-based schemes are less prone to implementation mistakes. Previous work has recognized the need for probabilistic encryption in order to achieve security. Rogaway has proved that nonce-based schemes are not only less likely to be misused, but they also satisfy stronger notions of security (see section 3.2.6).

Following [Rog04b], a nonce-based encryption scheme $n\mathcal{SE} = (\mathcal{K}, n\mathcal{E}, n\mathcal{D})$ consists of three algorithms: a probabilistic key generation algorithm $\mathcal{K}$, a deterministic encryption algorithm $n\mathcal{E}$ and a deterministic decryption one $n\mathcal{D}$. Similarly to the formalization previously given for symmetric encryption schemes, $\mathcal{K}$ takes as input a security parameter k and returns a key K. Syntax differs for $n\mathcal{E}$ and $n\mathcal{D}$. $n\mathcal{E}$ takes as input a secret key K with two user-provided inputs: a message $m \in \mathcal{M}$, where $\mathcal{M}$ is the message space, and a nonce $N \in \mathcal{N}$, where $\mathcal{N}$ is the nonce space. $n\mathcal{E}$ outputs a ciphertext $c \in \mathcal{C}$, where $\mathcal{C}$ is the ciphertext space. We write $c \longleftarrow n\mathcal{E}_{\mathsf{k}}(N, m)$. For the sake of completeness, we precise that $\mathcal{N}$, $\mathcal{M}$ and $\mathcal{C}$ are finite nonempty sets of strings. As for $n\mathcal{D}$, it takes as input a secret key K as well as a nonce $N \in \mathcal{N}$ and a ciphertext $c \in \mathcal{C}$ in order to output either a message $m \in \mathcal{M}$ or the special symbol $\perp$. We write $n\mathcal{D}_{\mathsf{k}}(N, c)$. Of course, we require that $n\mathcal{D}_{\mathsf{k}}(N, n\mathcal{E}_{\mathsf{k}}(N, m)) = m$ for all N and m.

Now, we present two nonce-based encryption schemes that use the CBC mode [Rog04b]. In both schemes, the underlying block cipher is applied over the nonce, and then the output is used as the initialization vector (IV) for the CBC mode. The two schemes are distinguished by the number of the required keys. The scheme CBC1 uses the same key to encrypt the nonce and the message, while the scheme CBC2 uses two independent keys for the two operations.

Definition 3.4. [*The One-Key Nonce-Based CBC Encryption Scheme (CBC1)*]
Let E be a block cipher of block size l . Let ivCBC$[E] = (\mathcal{K}\text{-ivCBC}, \mathcal{E}\text{-ivCBC}, \mathcal{D}\text{-ivCBC})$ be its associated IV CBC encryption scheme. Let K be a key. Given a nonce $N \in \mathcal{N}$ and a message $m \in \{0,1\}^{ln}$, we define the one-key CBC scheme CBC1$[E] = (\mathcal{K}\text{-CBC1}, n\mathcal{E}\text{-CBC1}, n\mathcal{D}\text{-CBC1})$:

CBC1 Encryption $n\mathcal{E}_{\mathsf{k}}$-CBC1$(N, m)$	**CBC1 Decryption** $n\mathcal{D}_{\mathsf{k}}$-CBC1$(N, c)$
1: $c_0 \longleftarrow E_{\mathsf{k}}(N)$	1: $c_0 \longleftarrow E_{\mathsf{k}}(N)$
2: $c \longleftarrow \mathcal{E}_{\mathsf{k}}$-ivCBC$(c_0, m)$	2: $m \longleftarrow \mathcal{D}_{\mathsf{k}}$-ivCBC$(c_0, c)$
3: **return** c	3: **return** m

Definition 3.5. [*The Two-Key Nonce-Based CBC Encryption Scheme (CBC2)*]
Let E be a block cipher of block size l. Let ivCBC$[E] = (\mathcal{K}\text{-ivCBC}, \mathcal{E}\text{-ivCBC}, \mathcal{D}\text{-ivCBC})$ be its associated IV CBC encryption scheme. Let K_1 and K_2 be two independent keys. Given a nonce $N \in \mathcal{N}$ and a message $m \in \{0,1\}^{ln}$, we define the two-key CBC scheme CBC2$[E] = (\mathcal{K}\text{-CBC2}, n\mathcal{E}\text{-CBC2}, n\mathcal{D}\text{-CBC2})$ as follows:

CBC2 Encryption $n\mathcal{E}_{\mathsf{k1},\mathsf{k2}}$-CBC2$(N, m)$	**CBC2 Decryption** $n\mathcal{D}_{\mathsf{k1},\mathsf{k2}}$-CBC2$(N, c)$
1: $c_0 \longleftarrow E_{\mathsf{k2}}(N)$	1: $c_0 \longleftarrow E_{\mathsf{k2}}(N)$
2: $c \longleftarrow \mathcal{E}_{\mathsf{k1}}$-ivCBC$(c_0, m)$	2: $m \longleftarrow \mathcal{D}_{\mathsf{k1}}$-ivCBC$(c_0, c)$
3: **return** c	3: **return** m

3.1.6 Encoding Schemes

Encryption schemes are used within protocols to protect the exchanged data. When defining a protocol, a plaintext message may have other associated data, such as a length field or a type field. Depending on how the protocol is constructed, this extra data are added before or after some cryptographic operation. The encoding scheme specifies what fields should be added and where they should be added. An encoding scheme $\mathcal{ES} = (\mathsf{Enc}, \mathsf{Dec})$ consists of an encoding function Enc, and a decoding function Dec. Decoding is necessary to remove additional fields and to perform several checks based on some fields. For instance, a length field must be checked to verify that the length is within a valid range. If a check fails, then this would usually result in an error message denoted by $\perp$.

Perhaps the most common type of encoding scheme is the padding scheme. As we described earlier, block ciphers only operate on messages of a fixed length. Therefore, a mode of operation might require inputs of length which is a multiple of this fixed block length. In order to operate on arbitrary length messages, we must *pad* the message before submitting it to the encryption algorithm. This padding operation must be reversible so that the receiver is able to strip off the padding after the message is decrypted.

3.1.7 Message Authentication Codes

The building blocks that we have defined so far only aim to provide confidentiality of messages. Another requirement for secure protocols is to ensure data authentication and integrity. Data authentication guarantees that the message was sent from a particular entity, while data integrity guarantees that the message has not been maliciously modified during transit.

In symmetric encryption schemes, one way to provide both data authentication and data integrity is to use a message authentication code (MAC). A sender generates a MAC tag over a message and sends it along with the message to the receiver. Once received, the receiver generates a new MAC tag using the received message and compares it with the received MAC tag. MACs can be constructed by various mechanisms. Below, we describe MAC built upon

block ciphers (e.g., CBC-MAC [BKR00]) and MAC built upon cryptographic hash functions (e.g. HMAC [KBC97]).

HMAC or Hash based MAC, uses a hash function to build a message authentication code. A hash function is deterministic and takes as input a message of arbitrary length to output a fixed-length message. Example of hash functions include the SHA (Secure Hash Algorithm), such as SHA2 and SHA3. To calculate a MAC on a message m, HMAC takes a hash function $H(.)$ and a key K to perform the following operation [KBC97]:

$$\mathrm{HMAC}_{\mathsf{k}}(m) = H((\mathsf{K} \oplus \mathtt{opad}) \,||\, H((\mathsf{K} \oplus \mathtt{ipad}) \,||\, m)).$$

Here, `opad` and `ipad` are two fixed padding constants.

Based on a block cipher, the CBC-MAC is a standardized message authentication code widely used in practice. It was first specified to be used with DES [oSN85]. Bellare, Kilian and Rogaway proved the security of the CBC-MAC for fixed message length [BKR00]. Therefore, several variants of CBC-MAC have been proposed for variable length messages. This includes OMAC [IK03] (and its NIST-recommended successor CMAC [Dwo01b]). OMAC, or One-Key CBC-MAC, is proved secure for arbitrary length messages.

Here, we only present the basic construction of CBC-MAC. Given a block cipher E of block size l and a message $m = m_1||...||m_n$, where $|m_i| = l$, the CBC-MAC[E] of m under key K is defined as τ_n, where

$$\tau_i = E_{\mathsf{k}}(m_i \oplus \tau_{i-1})$$

for $i = 1...n$ and $\tau_0 = \mathbf{0}^l$.

Formally, we define a message authentication scheme $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ to consist of a key generation algorithm $\mathcal{K}$, a tagging algorithm $\mathcal{T}$ and a verification algorithm $\mathcal{V}$. $\mathcal{K}$ takes as input a security parameter $k \in \mathbb{N}$ and returns a key K. The tagging algorithm $\mathcal{T}$ might be probabilistic and takes as input the key K and a message m to return a tag τ. The verification algorithm $\mathcal{V}$ is deterministic and takes as input the key K, the message m and a candidate tag τ to return a bit b. For correctness, we require that $\mathcal{V}_{\mathsf{k}}(m, \mathcal{T}_{\mathsf{k}}(m)) = 1$ for all m. We discuss the security properties required from a MAC in section 3.2.7.

3.1.8 Authenticated Encryption

Authenticated encryption (AE) is a symmetric encryption scheme that protects both data confidentiality and integrity. Integrity (authenticity) means that no adversary is able to produce new valid ciphertexts. This entails that encrypted data cannot be undetectably modified. Recently, the design of AE primitives has renewed interest, not least because of the currently running CAESAR competition [Ber15]. Generic composition [BN08] is the most popular approach for numerous security protocols, such as SSH, TLS and IPsec. This approach is about combining a confidentiality-providing encryption scheme together with a MAC. Nevertheless, the pursuit of more efficiency than that offered by these two-pass schemes has motivated the construction of dedicated AE designs, such as Galois Counter Mode (GCM) [MV03].

Generally, three composition methods are considered: Encrypt-and-MAC (EaM), MAC-then-Encrypt (MtE) and Encrypt-then-MAC (EtM). Given a symmetric encryption scheme $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$ and a message authentication code $\mathcal{MA} = (\mathcal{K}_m, \mathcal{T}, \mathcal{V})$, the composition methods are defined as follows given two independent keys K_1 and K_2:

1. *Encrypt-and-MAC*: In this method, encryption and message authentication are performed independently in parallel. That is, given a plaintext message m, the ciphertext is computed by appending c to τ, where $c \longleftarrow \mathcal{E}_{\mathsf{k}1}(m)$ and $\tau \longleftarrow \mathcal{T}_{\mathsf{k}2}(m)$. For the decryption operation,

c is first decrypted to recover m, and then the tag τ is verified. The message m is returned if $\mathcal{V}_{\mathsf{k2}}(m, \tau) = 1$, otherwise $\perp$ is returned.

2. *MAC-then-Encrypt*: Here, a MAC tag is first computed, then the message and the tag are encrypted together. That is, given a message m, the ciphertext c is computed as $\mathcal{E}_{\mathsf{k1}}(m||\tau)$, where $\tau = \mathcal{T}_{\mathsf{k2}}(m)$. For the decryption operation, c is first decrypted to obtain $m||\tau$. Then, m is returned if $\mathcal{V}_{\mathsf{k2}}(m, \tau) = 1$, otherwise $\perp$ is returned.

3. *Encrypt-then-MAC*: In this case, the message is first encrypted and then a MAC tag is computed over the result. That is, given a message m, the ciphertext is computed by appending c to τ, where $c \longleftarrow \mathcal{E}_{\mathsf{k1}}(m)$ and $\tau \longleftarrow \mathcal{T}_{\mathsf{k2}}(c)$. For the decryption operation, the tag τ is first verified. If $\mathcal{V}_{\mathsf{k2}}(c, \tau) = 1$, then the decryption of c is returned, otherwise, as before, $\perp$ is returned.

In some applications, it may be desirable to include some header data in the clear together with the ciphertext, but both the header and the ciphertext should be integrity-protected. Header data of this type is known as *associated data*. Schemes following the principle of generic composition are naturally adapted to protect header data. However, despite their efficiency, dedicated AE schemes are the ones suffering from the inability to deal with associated data. Rogaway was the first to formalize the problem of authenticated encryption with associated data (AEAD) [Rog02]. Since then, many AEAD schemes have been designed, for instance, Counter mode with CBC-MAC (CCM) mode [WHF03] and EAX [BRW04].

3.2 Provable Security

Provable security forms part of the large toolbox currently at the disposal of cryptographers. The techniques of provable security give cryptographers a means by which to formally analyze the security of schemes within a strict mathematical framework. Perhaps the earliest example of a mathematical proof of security of a cryptosystem was by Rabin in 1979 [Rab79], but it was not until the seminal work of Goldwasser and Micali [GM84] that a formal framework for provable security began to take shape. In their paper, Goldwasser and Micali introduced the fundamental and equivalent notions of semantic security and indistinguishability for public-key encryption. Despite being a quite new area of research, provable security has become one of the most important and studied topics in cryptography today. Dent gives a brief history of provable security within the context of public-key cryptography in [Den08].

The goal of provable security is to provide a rigorous *proof* that a construction, namely a cryptographic scheme, satisfies a given security definition characterizing a certain security requirement. Such proofs are important in the context of cryptography where there is an attacker who is actively trying to "break" some scheme. Proofs of security give an iron-clad guarantee, relative to the security definition, that no attacker will succeed. This is much better than taking an unprincipled or heuristic approach to the problem. Indeed, without a formal proof, we are only left with intuitions that no adversary can break the specified scheme. Experience has shown that intuition in computer security is disastrous. There are countless examples of unproven schemes that were broken, sometimes immediately and sometimes years after being standardized. In this thesis, we show, once again, that this is the case and how provable security is imperative for the design of complex systems.

Despite its clear advantages, the approach of provable security has received several criticisms. Bellare [Bel99] points out that the term provable security is slightly misleading. Security proofs do not actually prove the security of a given scheme, but instead just reduce it to the security of its underlying primitives or to the "hardness" of some mathematical problems. Another important point to notice is that these proofs of security only hold for a specific set of assumptions about the capabilities of the attacker. Thus, there might exist a class of attacks that could be carried out

in practice, even though the abstract model tells us the inverse. Koblitz and Menezes in [KM07] raise a stark warning by criticizing provable security. In this thesis, we emphasize that one must be aware of the context in order to interpret the results from provable security. Therefore, we do not only show theoretical weaknesses against cryptographic schemes, but we also extend their scope and apply them in practice to undermine the security of the whole underlying system.

3.2.1 Overview of Provable Security

Principles and Notations

One of the key contributions of modern cryptography has been the recognition that formal definitions of security are essential. Indeed, formal definitions give clear description of what should be required in order to achieve security. Therefore, the first step is to decide what actually means to be secure. There is no security definition that is better than all the other ones; the right one to use depends on the environment in which a cryptographic scheme is deployed.

In general, a security definition has two components: a security guarantee and a threat model. The security guarantee defines what the scheme is intended to prevent the attacker from doing, while the threat model describes what actions the attacker is assumed able to carry out. It is important to notice that a threat model solely concerns the abilities of the attacker and not her strategy. For a fixed security guarantee (aka security notion), a threat model is defined by precising the functions that can be called by the attacker. For this, the notation $\mathcal{A}^{\mathcal{O}}$ is used to denote the threat model in which the attacker (aka the *adversary*) $\mathcal{A}$ has access to the function (aka the oracle) $\mathcal{O}$.

More precisely, provable security starts by setting a *security goal*, and then it examines whether this goal is achieved by studying the probability that an *adversary wins* an *experiment* conducted by a *challenger*. This win condition can take many different forms, such as finding the decryption of a given ciphertext or correctly guessing which of two experiments is being played. During the experiment, the adversary might be allowed to access a set of *oracles* maintained by the said challenger. We call this experiment a *security model*. We associate to any adversary a number called her *advantage* that measures her success in winning the experiment. Henceforth, we say that a cryptographic scheme is *computationally secure* with respect to a given security model if all related polynomial-time adversaries have a *negligible* advantage. We write $\mathbf{Adv}^{\Pi}_{\mathcal{A},\mathcal{SM}}$ where $\mathcal{A}$ is an adversary who attempts to break the scheme Π regarding the security model $\mathcal{SM}$. For the sake of completeness, two points are clarified: polynomial-time adversaries and negligible advantage.

The era of "classical" cryptography begins with the concept of *perfect secrecy*. Perfect secrecy requires schemes to achieve their security goal even against an adversary with unlimited computational power. Despite its worth, perfect secrecy is unnecessarily strong and it fails short for practical purposes. Thus, the weaker (but sufficient) concept of *computational secrecy* was introduced. Computational secrecy incorporates two relaxations regarding perfect secrecy in order to go beyond the limitations of perfect secrecy. First, it allows for a small probability of failure. Indeed, adversaries can potentially succeed (i.e., security can potentially fail) with some very small probability. Second, it assumes adversaries with efficient computation. This means that given enough computational resources, an adversary might be able to violate security. Thus, if we can make the resources required to break the scheme larger than those available to any realistic attacker, then the scheme is unbreakable in practice. For instance, if the attacker needs 2^{128} attempts to break the scheme, then she will invest some hundreds of years to achieve her goal, which is considered as impractical in real-world systems.

Therefore, computational secrecy is defined in respect to *efficient algorithms* and *negligible* success probability. Below, we formally discuss these two notions.

Definition 3.6. [*Efficient Algorithm*]

An algorithm $\mathcal{A}$ is efficient if it runs in polynomial time. $\mathcal{A}$ runs in polynomial time if there exists a polynomial p, such that for every input $x \in \{0,1\}^*$ the computation of $\mathcal{A}(x)$ terminates within at most $p(|x|)$ steps. This means that the running time of a $\mathcal{A}$ is polynomially measured in terms of the length of its inputs. Here, we only consider adversaries who run in polynomial time with respect to the security parameter k.

Definition 3.7. [*Negligible Function*]
A function f is negligible if for every positive polynomial p and all sufficiently large values of n, we have that $f(n) < 1/p(n)$. Stated differently, let p be a positive polynomial, we say that f is negligible if it holds that:

$$\forall p, \quad \exists N \in \mathbb{N}, \quad \forall n \in \mathbb{N}, \ (n \geq N \Longrightarrow f(n) < 1/p(n)).$$

Proofs by Reduction

Unfortunately, most modern cryptographic constructions cannot be proved secure unconditionally. Indeed, proofs of security often rely on unproved assumptions, which involves to assume that some mathematical problem is *hard* (i.e. an arbitrary instance cannot be solved efficiently), or that some low-level cryptographic primitive (e.g., AES) is secure. This means that the related cryptographic construction is secure as long as some underlying problem is hard. Assumption-based proofs generally proceed by presenting an explicit *reduction* showing how to transform any efficient adversary $\mathcal{A}$ that succeeds in breaking the construction into an efficient algorithm $\mathcal{A}'$ that solves a problem assumed to be hard. To illustrate this important concept, we walk through a high-level outline of such a proof.

We begin by supposing that some problem X cannot be solved by any polynomial-time algorithm with non-negligible probability. We want to prove that some cryptographic construction Π is secure. The proof by reduction goes as follows (see Figure 3.1):

1. Fix some polynomial-time adversary $\mathcal{A}$ attacking Π.
2. Construct a polynomial-time algorithm $\mathcal{A}'$ that solves the problem X using $\mathcal{A}$ as a subroutine. In order to succeed, $\mathcal{A}'$ must *simulate* for $\mathcal{A}$ an instance of Π such that:
   a. $\mathcal{A}$ believes that she is interacting with Π.
   b. $\mathcal{A}'$ can transform an instance of X into an instance of Π in a polynomial time. $\mathcal{A}'$ can also transform a solution of Π into a solution of X in a polynomial time.
3. Taken together, 2(a) and 2(b) imply that if $\mathcal{A}$ succeeds in breaking Π, then $\mathcal{A}'$ solves an instance of the hard problem X. Therefore, we obtain an efficient algorithm $\mathcal{A}'$ solving X with non-negligible probability, which contradicts the initial assumption.
4. Thus, we conclude that no polynomial-time adversary $\mathcal{A}$ can succeed in breaking Π with non-negligible probability. For shorthand, Π is computationally secure.

Practice-Oriented Provable Security

Up until the mid 1990s, provable security was only associated with the field of public-key cryptography. Bellare and Rogaway were the first to take these techniques and apply them to the other type of cryptography, namely symmetric-key cryptography. Due to its relevance in practice, it is important that proofs of provable security be quantitative in the setting of symmetric cryptography. Traditionally, proofs in provable security took the following form: "the scheme is secure

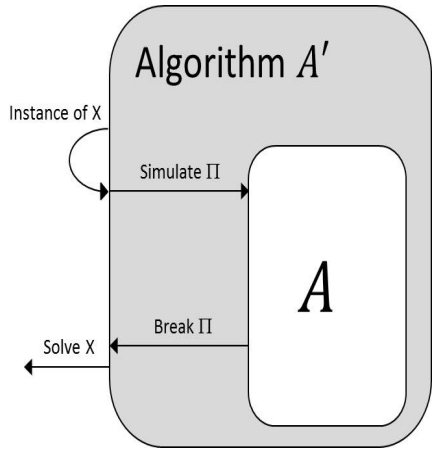


Figure 3.1: Overview of Proofs by Reduction

if some underlying problem is hard". A quantitative (aka concrete) proof [Bel99] provides an exact security analysis and draws precise bounds on the computational effort for the adversary in practice. Let us illustrate by an example of a concrete result. Assume we have a scheme S based upon some cryptographic primitive P. The scheme S is said to be secure against an adversary $\mathcal{A}$ running in time t with resources r, if P is secure against an adversary $\mathcal{A}'$ running in time t' with resources r'. Here, t and r are given in terms of t' and r'. In concrete proofs, the advantage of an adversary are given with respect to the security parameter (i.e., the key size), the block size (when considering block ciphers) and the number of queries made to the oracles. Now, we provide the formal definitions that are required to analyze schemes in the symmetric setting.

3.2.2 Functions and Permutations

As mentioned previously, block ciphers are one of the most important cryptographic primitives used in the construction of secure protocols. Therefore, we must formally model block ciphers in order to be able to use them in the context of provable security. One possible way of formalization is to consider block ciphers as pseudorandom function families. A function family is said to be pseudorandom if a function chosen uniformly at random from this family is indistinguishable from a function chosen uniformly at random from the set of *all* functions. We now provide a formal definition for a pseudorandom function family. In the following, a *distinguisher* is an algorithm that has access to an oracle and outputs a boolean; i.e., either 0 or 1. The distinguisher outputs 0 if it believes that the oracle output comes from a function chosen at random from the pseudorandom function family, and 1 otherwise (i.e. the oracle output comes from a function chosen at random from the set of all functions).

Definition 3.8. [*Pseudorandom Function Family (PRF)*]
Let $F = \{F_{\mathsf{k}} : \mathsf{K} \in \mathcal{K}(k)\}$ where F_{k} is a function mapping l-bit strings to l'-bit strings for each $\mathsf{K} \in \mathcal{K}(k)$. Let $\mathtt{Rand}^{l \to l'}$ be the set of functions from $\{0,1\}^l$ to $\{0,1\}^{l'}$. Let $b \in \{0,1\}$. Let D be a distinguisher who has access to the oracle $f_b(.)$ as defined in the following experiment:

Experiment $\mathbf{Exp}^{\text{prf}-b}_{F,\mathsf{D}}(k)$

1: $\mathsf{K} \xleftarrow{R} \mathcal{K}(k)$
2: $f_0 \xleftarrow{R} F_{\mathsf{k}},\ f_1 \xleftarrow{R} \mathtt{Rand}^{l \to l'}$
3: $b' \longleftarrow \mathsf{D}^{f_b(.)}$
4: **return** b'

The advantage of the distinguisher is defined as:

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{D},F}(k) = \left|\mathsf{Pr}[\mathbf{Exp}^{\mathrm{prf-0}}_{F,\mathsf{D}}(k) = 1] - \mathsf{Pr}[\mathbf{Exp}^{\mathrm{prf-1}}_{F,\mathsf{D}}(k) = 1]\right|$$

We say that F is a pseudorandom random function if for all polynomial-time distinguishers D, there is a negligible function negl such that: $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{D},F}(k) \leq \mathsf{negl}(k)$.

Distinguishing-based experiments, like the one described above, are quite popular in security models. For the sake of simplicity, hereafter, we describe such experiments by precising which oracle the adversary has access to. For instance, we denote $\mathbf{Exp}^{\mathrm{prf-0}}_{F,\mathsf{D}}(k)$ as $\mathsf{D}^{F_{\mathsf{k}}(.)}(k)$ and $\mathbf{Exp}^{\mathrm{prf-1}}_{F,\mathsf{D}}(k)$ as $\mathsf{D}^{f(.)}$. where $F_{\mathsf{k}}(.)$ is taken over uniform choice of $\mathsf{K} \in \mathcal{K}(k)$ and $f(.)$ is taken over uniform choice of $f \in \mathtt{Rand}^{l \to l'}$. Thus, we can state the definition of PRF differently.

F is a pseudorandom function if for all polynomial-time distinguishers D, there is a negligible function negl such that:

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{D},F}(k) = \left|\mathsf{Pr}[\mathsf{D}^{F_{\mathsf{k}}(.)}(k) = 1] - \mathsf{Pr}[\mathsf{D}^{f(.)}(k) = 1]\right| \leq \mathsf{negl}(k)$$

where F_{k} and $f(.)$ are as defined above.

Recall from section 3.1.2 that block ciphers are defined as a family of permutations. Below, we present the definition of a pseudorandom permutation.

Definition 3.9. [*Pseudorandom Permutation Family (PRP)*]
Let $F = \{F_{\mathsf{k}} : \mathsf{K} \in \mathcal{K}(k)\}$ where F_{k} is a permutation over $\{0,1\}^l$ for each $\mathsf{K} \in \mathcal{K}(k)$. Let $\mathtt{Perm}^l$ be the set of permutations over $\{0,1\}^l$. F is a pseudorandom permutation if for all polynomial-time distinguishers D, there is a negligible function negl such that:

$$\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{D},F}(k) = \left|\mathsf{Pr}[\mathsf{D}^{F_{\mathsf{k}}(.)}(k) = 1] - \mathsf{Pr}[\mathsf{D}^{f(.)}(k) = 1]\right| \leq \mathsf{negl}(k)$$

where $F_{\mathsf{k}}(.)$ is taken over uniform choice of $\mathsf{K} \in \mathcal{K}(k)$ and $f(.)$ is taken over uniform choice of $f \in \mathtt{Perm}^l$.

Similarly to the relation between permutations and functions, namely all permutations are also functions, it is proven that a pseudorandom permutation is also a pseudorandom function.

Result 3.1. [*PRPs are PRFs*] *[BDJR97]*
Let $F = \{F_{\mathsf{k}} : \mathsf{K} \in \mathcal{K}(k)\}$ where F_{k} is a permutation over $\{0,1\}^l$ for each $\mathsf{K} \in \mathcal{K}(k)$. Consider any PRF-distinguisher D attacking F and asking q queries. Then, we can construct a PRP-distinguisher D' such that:

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{D},F}(k) \leq \mathbf{Adv}^{\mathrm{prp}}_{\mathsf{D}',F}(k) + q^2/2^{l-1}$$

3.2.3 Threat Models for Symmetric Encryption

In the context of symmetric encryption, there exist several attack models. In this thesis, we only consider the most standard ones which, in order of increasing power of the attacker, are:

- **Eavesdropper:** This is the most basic attack. It refers to a scenario in which the adversary attempts to deduce some information about plaintexts by just observing their corresponding ciphertexts.
- **Chosen-Plaintext Attack (CPA):** In this attack, the adversary can obtain plaintext/-ciphertext pairs for plaintexts of *her* choice. Then, she tries to deduce some information about the underlying plaintext of some *other* ciphertext produced by the same key. We say that the adversary has access to an encryption oracle.

- **Chosen-Ciphertext Attack (CCA):** In addition to her chosen-plaintext capability, the adversary is able to obtain the decryption of ciphertexts of her choice. Once again, the aim of the adversary is to learn some information about the plaintext of some other ciphertext. We say that the adversary has access to both an encryption and a decryption oracle.

The first type of attack is the easiest to carry out. The only thing that the adversary needs to do is to listen on the communication channel over which ciphertexts are sent. In the latter two attacks, the adversary is assumed to be able to obtain encryption and decryption of plaintexts/-ciphertexts of her choice. At first glance, this might seem strange. We present practical attacks under these threat models in this thesis.

3.2.4 Confidentiality of Symmetric Encryption

Encryption was firstly defined to provide *confidentiality*. The concept of confidentiality captures several meanings. Some believe that a formal definition is not needed, since everyone has an intuitive idea of what confidentiality means. However, this is a common mistake, since no valid proof of security would be possible without a precise definition. As mentioned before, a given cryptographic construction is secure if it satisfies its definition.

The most natural way to see confidentiality is that no adversary can learn any bit of information about the underlying plaintext of a given ciphertext. This idea was formalized under the notion of *semantic security* that was first introduced by Goldwasser and Micali [GM84] and later refined by Goldreich [Gol04]. Semantic security is a computational-complexity version of *perfect secrecy* introduced by Shannon in the context of information-theoretic security [Sha49]. Loosely speaking, an encryption scheme is semantically secure if it is infeasible to learn anything about the plaintext from the ciphertext. Put it differently, nothing can be gained from the ciphertext in the sense that whatever can be learned about a plaintext from its ciphertext can also be learned without it. Although the definition of semantic security might look simple, it is hard to use it to prove the security of an encryption scheme. Therefore, a more convenient notion of confidentiality, namely *indistinguishability*, was defined by Goldwasser et al. [GM84].

Indistinguishability requires that no polynomial-time adversary be able to distinguish the encryption of two plaintexts (of the same length). Indistinguishability might seem odd and counter-intuitive, but experience shows that it is more convenient to use for security proofs. Moreover, despite their apparent difference, authors in [GM84] prove the equivalence between it and semantic security. Bellare [Bel99] proposes several models to define indistinguishability in the setting of symmetric encryption. In this thesis, we consider the strongest model, that is *Left-or-Right Indistinguishability (IND).*

Following the principles of provable security, IND was formalized by [BDJR97] as an experiment. Let $\mathcal{E}_{\mathsf{k}}(\mathcal{LR}(.,.,b))$, where $b \in \{0,1\}$, be a left-or-right oracle for an encryption scheme $\mathcal{E}_{\mathsf{k}}$. This oracle takes two messages as input, m_0 and m_1, and returns $c \longleftarrow \mathcal{E}_{\mathsf{k}}(m_b)$. In IND, the adversary has access to the oracle $\mathcal{E}_{\mathsf{k}}(\mathcal{LR}(.,.,b))$. She submits queries of the form (m_0, m_1) , where $|m_0| = |m_1|$, and receives a ciphertext. The adversary wins if she can efficiently guess the boolean b, i.e. which message was encrypted. If all adversaries cannot succeed with probability better than a random guess, then the encryption scheme is called secure in the sense IND-ATK, where ATK represents the attack model. From section 3.2.3, we see that ATK $\in \{\text{CPA}, \text{CCA}\}$.

Definition 3.10. [*Left-or-Right Indistinguishability (IND)*].
Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. Let ATK $\in \{\text{CPA}, \text{CCA}\}$. $\mathcal{SE}$ is IND-ATK secure if for all polynomial-time distinguishers D, there is a negligible function negl:

$$\mathbf{Adv}^{\text{ind-atk}}_{\mathsf{D},\mathcal{SE}}(k) = \left|\mathsf{Pr}[\mathsf{D}^{\mathcal{E}_{\mathsf{k}}(\mathcal{LR}(.,.,0)),\mathcal{O}}(k) = 1] - \mathsf{Pr}[\mathsf{D}^{\mathcal{E}_{\mathsf{k}}(\mathcal{LR}(.,.,1)),\mathcal{O}}(k) = 1]\right| \leq \mathsf{negl}(k)$$

where $\mathcal{O}$ is an oracle that depends on the threat model: $\mathcal{O} = \mathcal{D}_{\mathsf{k}}(.)$ when ATK = CCA, and $\mathcal{O} = \epsilon(.)$, where $\epsilon(.)$ is the identity function, when ATK = CPA.

Two restrictions are set on the distinguisher D in order to avoid trivial attacks. First, the messages submitted by D to the encryption oracle $\mathcal{E}_{\mathsf{k}}(\mathcal{LR}(.,.,b))$ must be of same length. Second, D is prohibited from querying the oracle $\mathcal{D}_{\mathsf{k}}(.)$ on a ciphertext output by the encryption oracle when ATK = CCA.

3.2.5 Integrity of Symmetric Encryption

Encryption schemes do not only protect confidentiality, they can also protect integrity. In the symmetric-key settings, integrity (authenticity), means, loosely speaking, that only valid parties possessing the secret key K are able to produce valid ciphertexts; i.e. whose decryption does not give $\perp$. Symmetric encryption schemes in general do not protect the integrity of messages. For example, the CBC or the CTR modes do not provide integrity, since they never return $\perp$.

Throughout this thesis, we consider two notions of integrity: integrity of ciphertext (INT-CTXT) [BN08] and ciphertext unforgeability (CUF-CPA) [Kra01]. In both notions, the adversary is challenged to create a valid ciphertext forgery. To win the game, the adversary must submit a valid ciphertext to the decryption oracle that has not previously been output by the encryption oracle. INT-CTXT and CUF-CPA differ on how many times the adversary is allowed to call the decryption oracle. Indeed, contrary to INT-CTXT, the adversary in CUF-CPA can only call the decryption oracle once to verify her attempted forgery. Formal definitions are given below.

Definition 3.11. [*Integrity of Ciphertext (INT-CTXT)*]
Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. Let $\mathcal{A}$ be a polynomial-time adversary. Let S be the list of all ciphertexts generated by the adversary queries to $\mathcal{E}_{\mathsf{k}}(.)$. For $k \in \mathbb{N}$, the following experiment is defined:

Experiment $\mathbf{Exp}^{\text{int-ctxt}}_{\mathcal{SE},\mathcal{A}}(k)$

$\mathsf{K} \xleftarrow{R} \mathcal{K}(k)$
if $c \longleftarrow \mathcal{A}^{\mathcal{E}_{\mathsf{k}}(.),\mathcal{D}_{\mathsf{k}}(.)}$ such that
$\mathcal{D}_{\mathsf{k}}(c) \neq \perp$ and $c \notin \mathsf{S}$ **then**
  return 1
else
  return 0
end if

$\mathcal{SE}$ is INT-CTXT secure if for all polynomial-time adversaries $\mathcal{A}$, there is a negligible function negl such that:

$$\mathbf{Adv}^{\text{int-ctxt}}_{\mathcal{A},\mathcal{SE}}(k) = \mathsf{Pr}[\mathbf{Exp}^{\text{int-ctxt}}_{\mathcal{SE},\mathcal{A}}(k) = 1] \leq \mathsf{negl}(k)$$

Considering their definitions, confidentiality and integrity are two different security goals. However, it is best practice to always ensure both of them. Moreover, it turns out that integrity is essential in many applications where confidentiality is required. Indeed, it can be sometimes hard to directly prove that a scheme provides confidentiality against a power adversary, i.e. IND-CCA. Therefore, numerous security proofs break down the security model IND-CCA into other weaker ones. It was proved that any encryption scheme providing IND-CPA together with INT-CTXT is also IND-CCA secure.

Result 3.2. [*IND-CPA* $\wedge$ *INT-CTXT* $\Longrightarrow$ *IND-CCA*] *[BN08]*
Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. For any polynomial-time IND-CCA adversary $\mathcal{A}$ attacking $\mathcal{SE}$, we can construct two adversaries $\mathcal{B}$ and $\mathcal{C}$ such that:

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{A},\mathcal{SE}}(k) \leq \mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{B},\mathcal{SE}}(k) + 2\mathbf{Adv}^{\text{int-ctxt}}_{\mathcal{C},\mathcal{SE}}(k)$$

Definition 3.12. [*Ciphertext Unforgeability (CUF-CPA)*]
Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. Let $\mathcal{A}$ be a polynomial-time adversary. Let S be the list of all ciphertexts generated by the adversary queries to $\mathcal{E}_{\mathsf{k}}(.)$. For $k \in \mathbb{N}$, the following experiment is defined:

Experiment $\mathbf{Exp}^{\text{cuf-cpa}}_{\mathcal{SE},\mathcal{A}}(k)$

```
K ←R K(k)
c ← A^{E_k(.)}
if D_k(c) ≠ ⊥ and c ∉ S then
    return 1
else
    return 0
end if
```

$\mathcal{SE}$ is CUF-CPA secure if for all polynomial-time adversaries $\mathcal{A}$, there is a negligible function negl such that:

$$\mathbf{Adv}^{\text{cuf-cpa}}_{\mathcal{A},\mathcal{SE}}(k) = \mathsf{Pr}[\mathbf{Exp}^{\text{cuf-cpa}}_{\mathcal{SE},\mathcal{A}}(k) = 1] \leq \mathsf{negl}(k)$$

Despite being a weaker notion, Paterson et al. prove the following relation between CUF-CPA and INT-CTXT.

Result 3.3. [*CUF-CPA* $\Longrightarrow$ *INT-CTXT*] *[PW12]*
Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. Consider any INT-CTXT adversary $\mathcal{A}$ attacking $\mathcal{SE}$ and making q_d queries to the decryption oracle. Then, we can construct a CUF-CPA adversary $\mathcal{B}$ such that:

$$\mathbf{Adv}^{\text{int-ctxt}}_{\mathcal{A},\mathcal{SE}}(k) \leq q_d \mathbf{Adv}^{\text{cuf-cpa}}_{\mathcal{B},\mathcal{SE}}(k)$$

We have up to this point only defined classical security models for symmetric encryption schemes. Now, we consider other security models of other cryptographic primitives.

3.2.6 Indistinguishability from Random Bits

Symmetric encryption schemes satisfying left-or-right indistinguishability (IND) often satisfy a stronger notion of indistinguishability known as *indistinguishability from random bits* (IND\$). This notion was put forward by Rogaway in [Rog04b]. Under this notion, a scheme is secure if ciphertexts cannot be distinguished from random strings of the same length. Note that IND\$ implies IND when the length of ciphertexts depends only on the message length and not on the message itself. For instance, let m be a message containing only one byte x. Then, the length of the produced ciphertext depends on m's length (i.e. 1) and not on its content (i.e. x). Surprisingly, proving IND\$ security is often simpler than proving IND security for many encryption schemes. In what follows, we provide the definition of IND\$ under chosen-plaintext attacks (IND\$-CPA)[4]:

Definition 3.13. [*Indistinguishability from Random Bits under CPA (IND\$-CPA)*]
Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. $\mathcal{SE}$ is IND\$-CPA secure if for all polynomial-time distinguishers D, there is a negligible function negl such that:

$$\mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{D},\mathcal{SE}}(k) = \left|\mathsf{Pr}[\mathcal{D}^{\mathcal{E}_{\mathsf{k}}(.)}(k) = 1] - \mathsf{Pr}[\mathcal{D}^{\mathcal{R}(.)}(k) = 1]\right| \leq \mathsf{negl}(k)$$

[4] An analogous definition can be given for IND\$ under chosen-ciphertext attacks (IND\$-CCA)

where $\mathcal{R}(.)$ is a function that returns some uniformly random bits of the same length as the encrypted message.

We underline two points from the above definition. First, IND\$-CPA just describes the idea that the encryption result should look like a string of random bits. Second, the experience of the distinguisher $\mathcal{D}$ gives her access to one of two oracles. When $\mathcal{D}$ submits a query m, she receives the output of an oracle and her goal is to determine which oracle she is communicating with. In IND\$, the two oracles are (1) the encryption algorithm $\mathcal{E}_k(.)$ of $\mathcal{SE}$ that returns the real ciphertext of the query ($c \longleftarrow \mathcal{E}_{\mathsf{k}}(m)$), and (2) a random oracle $\mathcal{R}(.)$ that on any input returns some random bits of length $|c|$.

3.2.7 Strong Unforgeability of MAC

We now define the default notion of security for message authentication codes. The intuitive idea behind the definition is that no polynomial-time adversary should be able to generate a new valid tag for a message. As with any security definition, we should unambiguously define the power of the adversary (i.e., threat model) and the conditions to be met in order to break the scheme.

Concerning the threat model, the adversary (or the forger) is allowed to request the MAC tag for any message of her choice. Formally, for a MAC scheme $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$, the forger is given access to the tagging algorithm $\mathcal{T}_{\mathsf{k}}(.)$ which she can use to get the tag of any message m. In addition, she has access to the verification algorithm $\mathcal{V}_{\mathsf{k}}$ to check the validity of her forgery.

As for the security goal, the forger breaks $\mathcal{MA}$ if she outputs a pair of a message and a tag (m, τ) such that: (1) τ is a valid tag on the message m (i.e., $\mathcal{V}_{\mathsf{k}}(m, \tau) = 1$), and (2) τ has never been generated by $\mathcal{T}_{\mathsf{k}}(.)$ as a tag of m. The second condition is required because it is always possible for the forger to just copy a pair of a message and its tag that was previously sent by a legitimate party. Of course, such a pair is valid and satisfies the first condition, which implies that MAC schemes do not protect against *replay attacks*. This does not mean that replay attacks are not a security concern. We will have more to say about them in chapter 5.

Definition 3.14. [*Strong Unforgeability for MAC (SUF-CMA)*]
Let $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a message authentication scheme. Let $\mathcal{F}$ be a polynomial-time forger with access to the oracles $\mathcal{T}_{\mathsf{k}}(.)$ and $\mathcal{V}_{\mathsf{k}}(.,.)$. Let S be the list of all inputs/outputs submitted and returned by the oracle $\mathcal{T}_{\mathsf{k}}(.)$. For $k \in \mathbb{N}$, the following experiment is defined:

Experiment $\mathbf{Exp}^{\text{suf-cma}}_{\mathcal{MA},\mathcal{F}}(k)$

1: $\mathsf{K} \xleftarrow{R} \mathcal{K}(k)$
2: $(m, \tau) \longleftarrow \mathcal{F}^{\mathcal{T}_{\mathsf{k}}(.), \mathcal{V}_{\mathsf{k}}(.,.)}$
3: **if** $\mathcal{V}_{\mathsf{k}}(m, \tau) = 1$ and $(m, \tau) \notin \mathsf{S}$ **then**
4: **return** 1
5: **else**
6: **return** 0
7: **end if**

$\mathcal{MA}$ is SUF-CMA secure if for all polynomial-time forgers $\mathcal{F}$, there is a negligible function negl such that:

$$\mathbf{Adv}^{\text{suf-cma}}_{\mathcal{F},\mathcal{MA}}(k) = \mathsf{Pr}[\mathbf{Exp}^{\text{suf-cma}}_{\mathcal{MA},\mathcal{F}}(k) = 1] \leq \mathsf{negl}(k)$$

3.2.8 Results for Generic Compositions

As mentioned in section 3.1.8, an encryption scheme and a MAC scheme can be combined to provide both confidentiality and integrity. Generally speaking, there are three methods for

generic composition: Encrypt-and-MAC (EaM), MAC-then-Encrypt (MtE) and Encrypt-then-MAC (EtM). These three compositions were formally analyzed by Bellare and Namprempre in [BN08]. We now give a short summary of their results. Below, we assume that each composite scheme $\mathcal{AE}[\mathcal{SE}, \mathcal{MA}] = (\mathcal{K}, \mathcal{E}\text{-}\mathcal{AE}, \mathcal{D}\text{-}\mathcal{AE})$ was constructed by combining the encryption scheme $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$ and the MAC scheme $\mathcal{MA} = (\mathcal{K}_m, \mathcal{T}, \mathcal{V})$.

Result 3.4. [*EaM is not Generically IND-CPA Secure*] Given an IND-CPA secure symmetric encryption scheme $\mathcal{SE}$, we can construct an SUF-CMA secure message authentication scheme $\mathcal{MA}$ such that the composite scheme $\mathcal{AE}[\mathcal{SE}, \mathcal{MA}]$ formed by following the generic method Encrypt-and-MAC is not IND-CPA secure.

Result 3.5. [*EaM is not Generically INT-CTXT Secure*] Given an SUF-CMA secure message authentication scheme $\mathcal{MA}$, we can construct an IND-CPA secure symmetric encryption scheme $\mathcal{SE}$ such that the composite scheme $\mathcal{AE}[\mathcal{SE}, \mathcal{MA}]$ formed by following the generic method Encrypt-and-MAC is not INT-CTXT secure.

Result 3.6. [*MtE is not Generically IND-CCA Secure*] Given an SUF-CMA secure message authentication scheme $\mathcal{MA}$, we can construct an IND-CPA secure symmetric encryption scheme $\mathcal{SE}$ such that the composite scheme $\mathcal{AE}[\mathcal{SE}, \mathcal{MA}]$ formed by following the generic method MAC-then-Encrypt is not IND-CCA secure.

Result 3.7. [*EtM is Generically IND-CCA Secure*] Given an IND-CPA symmetric encryption scheme $\mathcal{SE}$ and an SUF-CMA secure message authentication scheme $\mathcal{MA}$, the composite scheme $\mathcal{AE}[\mathcal{SE}, \mathcal{MA}]$ formed by following the generic method Encrypt-then-MAC is both IND-CPA and INT-CTXT secure.

Part II

Formal Security Analysis of Two Real-World Systems

Chapter 4

Breaking Into the Android KeyStore

"Ethical hacking!! Is it not an oxymoron?"

– Ankala V Subbarao

Contents

4.1 Responsible Disclosure **52**
4.2 Introduction **52**
4.2.1 Overview of our Attack 53
4.2.2 Context in Trusted Computing 53
4.3 Background **53**
4.4 Related Work **54**
4.5 Hash-then-Encrypt Security Results **56**
4.5.1 Integrity Notions 56
4.5.2 Hash-then-Encrypt 56
4.5.3 Hash-then-Encrypt is IND-CPA Secure 56
4.5.4 Hash-then-CBC-Encrypt 57
4.5.5 Hash-then-CTR-Encrypt 59
4.6 The Android KeyStore **60**
4.6.1 KeyStore Service 61
4.6.2 The Keys Formats 61
4.7 Attacking the Android KeyStore **62**
4.7.1 Threat Model 62
4.7.2 The Forgery Attack 62
4.7.3 The Undetected Malware 63
4.8 Discussion and Recommendations **66**
4.9 Conclusion **67**

The Android KeyStore is a system service whose purpose is to shield users cryptographic keys. The KeyStore protects the integrity and the confidentiality of keys by using a particular encryption scheme. In this chapter, we analyze the security of this encryption scheme using the paradigm of provable security. Our results are twofold. First, we formally prove that the used encryption scheme does not provide integrity, which means that an adversary is able to undetectably modify the stored keys. Second, we exploit this flaw to define a forgery attack breaching the security guaranteed by the KeyStore. In particular, our attack allows a malicious application to make mobile apps to unwittingly perform secure protocols using weak keys. The threat is concrete: the attacker goes undetected while compromising the security of users. The findings of this chapter

were published in [ST16a] that was presented at the 21st *European Symposium on Research in Computer Security (ESORICS 2016).*

4.1 Responsible Disclosure

We communicated our findings to Google in January 2016 using the public bug tracker of the Android Open Source Project (AOSP) [And16b]. Our notification was noted under the label `AndroidID-26804580`. In March 2016, the Android security team has acknowledged the attack presented in this chapter and confirmed that the broken encryption scheme is planned for removal.

4.2 Introduction

The KeyStore is a component of paramount importance in the Android architecture. In fact, it has been updated continuously since Android 4.0, namely since October 2011. Beginning with Android 4.3 (aka Jelly Bean) released in 2012, the KeyStore allows third-party applications to generate, use and store their cryptographic keys. Once keys are in the KeyStore, they are protected from extraction using two security measures. First, the keys inside the KeyStore cannot be directly manipulated by their own application. Indeed, when an application needs to use its key, it asks the KeyStore to execute the required cryptographic operation with its associated key. Thus, keys never enter the application process, and therefore they are protected even when the application is compromised. Second, the keys are stored on the device in encrypted form. Hence, keys are not exposed even when the attacker has access to the internal storage of the mobile.

Multiple implementations exist for the KeyStore. Each mobile manufacturer might choose not to use the default implementation provided by Google and define its specific one. Each implementation includes its own security measure for the KeyStore. In this chapter, we focus solely on the security measure concerning the secure storage of keys, since it is the only one that is not implementation-dependent. Our goal is to analyze this security measure and to verify whether it indeed accomplishes its security objectives. Due to its great importance, we require rigor when it is about the KeyStore, and therefore we rely on the paradigm of provable security to perform our analysis.

As mentioned above, the KeyStore protects the integrity and the confidentiality of its keys by storing them in encrypted form using authenticated encryption (AE). For some reason, the scheme in use is particular and does not follow any standardized or provably secure construction. Its idea is simple: the message (representing the stored key) is appended to its MD5 hash value before encrypting it with CBC mode. Henceforth, we call this AE scheme *Hash-then-Encrypt* using the CBC mode (or Hash-then-CBC-Encrypt for shorthand).

At the first look, Hash-then-Encrypt is a lightweight mode that has many advantages over other popular AE schemes. It is more efficient than those based on the generic composition approach [BN08], since the message is needed not to be processed twice. In addition, it is much simpler to implement compared to others. Therefore, it might seem to be the most fitting scheme to implement inside mobile devices for the protection of users keys.

In this chapter, we show that intuition often goes wrong when security is concerned. We start by proving that for the two encryption modes CBC and CTR, the AE scheme **Hash-then-Encrypt** does not provide integrity **regardless of the used hash function**. To this end, we show that it does not satisfy the two notions of integrity: integrity of ciphertext (INT-CTXT) and ciphertext unforgeability (CUF-CPA). Then, we present a selective forgery attack where an adversary exploits this weakness to substantially reduce the length of the symmetric keys stored inside the KeyStore.

4.2.1 Overview of our Attack

The basic idea behind our attack scenario against the KeyStore is quite simple and can be explained as follows. The scenario starts when the mobile user installs an application which entrusts the KeyStore with its symmetric key. Using the integrity flaw of Hash-then-Encrypt, the attacker silently shortens the symmetric key associated to the installed application. For instance, she can transform 256-bit HMAC keys into 32-bit ones. The KeyStore will not detect this forgery and will keep on using the shortened key as it was the original key generated for the application. This allows a malicious third party who controls the network to break any secure protocol based on this weak key. Indeed, a simple exhaustive attack is enough to recover the application key even when secure protocols are used. Such an attack constitutes a real threat for two reasons: (1) it happens undetected, since no communication is required between the attacker and the malicious third party; and (2) the victim (mobile user) is lulled into a false sense of security because she believes that her application is well protected using a secure protocol with a strong key.

4.2.2 Context in Trusted Computing

This chapter is about to analyze the security guaranteed by the Android KeyStore. Being a component of great value, we might believe that all the necessary efforts have been made to properly design its security measures. Therefore, some may tend to *trust* the KeyStore and cast no doubt about its security. Here, we approach this problem of trust in a different manner. Indeed, we use the tools of provable security to formally investigate the soundness of the KeyStore design.

Our work brings to light an interesting fact: security in modern systems still does not withstand a simple cryptanalysis. This is mainly due to the fact that, unfortunately, system designers are usually tempted to use cryptographic schemes not for their proved security but for their straightforwardness and flexibility to meet special needs. We show that, once again, this a bad practice when designing a complex system, since such schemes are often vulnerable and can result in severe consequences for the whole underlying system.

The particularity of our work is that we use advanced security notions, such as indistinguishability, in order to compromise a system like Android. Our attack demonstrates that no theoretical weakness concerning the security of a cryptographic scheme should be ignored. Indeed, it could be utilized in one way or another to break the whole system. We thus show that the scope of these notions extends beyond theory and how they are relevant for system designers. We advocate the shift onto *provably secure cryptography* in order to prevent potential vulnerabilities that will be hard to find inside a complex system.

4.3 Background

In order to better understand this chapter, readers are required to be familiar with some background information:

- The architecture of the KeyStore (section 2.3.4);
- Encryption schemes and modes of operation (section 3.1.4);
- Authenticated encryption schemes (section 3.1.8);
- The principles of provable security and the proofs of reduction (section 3.2.1);
- Indistinguishability under chosen-plaintext/ciphertext attacks (section 3.2.4);
- Integrity of ciphertexts and ciphertext unforgeability (section 3.2.5);
- Types of forgery attacks (described below).

Forgery Attacks

When integrity is concerned, the adversary is often called a *forger* and successful attacks are called *forgery*. We distinguish three types of forgeries depending on the adversary capability:

1. ***Universal forgery***: forgery is possible for any given message;
2. ***Selective forgery***: forgery is possible for any given message chosen before the attack;
3. ***Existential forgery***: forgery is possible for at least one message.

4.4 Related Work

The two main research directions that our work targets is the KeyStore security and the analysis of authenticated encryption schemes. Here, we discuss the most relevant related work and argue how our work is distinguished from others.

KeyStore Security

Encryption in mobile devices is increasingly becoming a topic of utmost importance. Teufl et al. have thoroughly analyzed the encryption components of Android in [TFH+14]. This concerns both full disk encryption and credential storage. Authors only provide a descriptive study of the two systems. However, no cryptanalysis of the presented cryptographic schemes is given. Works in [GM14, MS13] highlight the severity of physical attacks, such as cold boot, against Android's disk encryption. Similar attacks might be done against the KeyStore to recover the master key ensuring secure storage. The primary limitation of these attacks is that they require a physical access to the targeted mobile devices. Therefore, the real threat is constrained.

As for secure credential storage, authors in [ZWWJ15] show that app developers tend to implement their own mechanisms to store credentials. They underline the prevalence of flawed solutions by designing a tool capable of automatically identifying and retrieving app credentials. Developers are thus urged to use the security services proposed by the Android system itself, that is KeyStore. The different implementation flavors, software-based and hardware-based, of the KeyStore are subjected to close scrutiny in [CdRP14]. The investigation involves how an attacker is able to compromise the different access controls to the stored keys. Authors conclude that an application with root privileges can have access to keys of other applications even when keys are bound to a secure hardware like Trusted Execution Environment. Their presented study assumes that all the cryptographic algorithms were properly built and implemented, which is proved not to be true in [HD14]. Hay et al. exploit a buffer overflow vulnerability that enables the execution of an arbitrary code inside the KeyStore process.

To the best of our knowledge, we present the first cryptanalysis-based attack against the KeyStore. A particularity of our work is the approach that we follow to mount the final attack. Basically, our approach consists of three steps: (1) we started by analyzing the Android code to find out what cryptographic scheme used for secure storage; (2) Then, we investigated the security of the encryption scheme using provable security; (3) Finally, we figured out how to exploit the discovered weakness to perform a practical attack against the Android system.

Authenticated Encryption

Recall that authenticated encryption is a symmetric encryption scheme that protects both data confidentiality and integrity (authenticity). In order to design such schemes, generic composition is the most popular approach for numerous security protocols. This approach is about combining a confidentiality-providing encryption scheme together with a MAC scheme. Nevertheless,

the pursuit of more efficiency than that offered by these two-pass schemes has motivated the construction of dedicated AE designs.

It turns out that designers do not only strive for efficiency, but also for implementation simplicity. Therefore, authenticity obtained from *Encryption-with-Redundancy* (EwR) has long been attractive. In such a paradigm, encryption consists of computing some public function h over the message m to get a checksum $\sigma = h(m)$. Then, $m||\sigma$ is encrypted and returned. As for decryption, the ciphertext is decrypted to get $m||\sigma$ and then the equality $\sigma = h(M)$ is verified. If the equality holds, the decryption algorithm returns the message m, otherwise it returns $\perp$.

Several of these schemes have been partially or fully broken [Mit05a, Mit13]. A generic attack attributed to Wagner on a large class of EwR using the CBC mode is described in [Pre98]. An and Bellare in [AB01] formally prove that this AE scheme does not generically guarantee security.

Some might argue that Hash-then-Encrypt (HtE) is just a special case of EwR, where the checksum function h is a hash function. However, we argue that this is not true. Indeed, the checksum is appended at the end of the message ($m||\sigma$) in EwR, while the hash value is appended at the beginning of the message ($\sigma||M$) in HtE. Thus, generic attacks against EwR, Wagner's for instance, are easier to apply than those against HtE. This is due to the fact that the former typically requires to remove the last block of ciphertexts, and for many schemes, the decryption of the first blocks does not depend on the last ones (e.g. CBC and CTR). For instance, in CBC mode, the decryption algorithm decrypts the blocks of ciphertexts sequentially, and thus the first blocks can be decrypted even when the last ones have been removed. We notice that the inverse is not true. Indeed, removing the first blocks would prevent the subsequent blocks from being correctly decrypted. One can easily verify this for the CBC and the CTR modes. As a result, the generic attack of Wagner against EwR that consists of removing the last block of ciphertext is not trivially adaptable against HtE, since HtE instead requires to remove the first block.

Now, let us discuss the proof of [AB01]. An and Bellare provide a proof showing that the construction EwR does not always guarantee security. In fact, we can construct a secure encryption scheme such that the associated EwR does not offer integrity. A similar result related to MAC-then-Encrypt (MtE) is given in [BN08] (see result 3.6). In order to avoid misinterpretation, we emphasize that these results only imply that such constructions are not generically secure. *Generic security* is a strong notion of security. It rejects as "unsafe" any construction for which there exists a single counterexample in which the soundness of the underlying schemes does not constitute a sufficient condition to guarantee security. Indeed, the proof of Bellare consists of providing a counterexample, i.e. a particular MtE scheme that it is not IND-CCA secure although its encryption and MAC algorithms are respectively IND-CPA and SUF-CMA secure. The proof is only applicable for a specific class of IND-CPA encryption schemes whose ciphertexts can be modified without changing their corresponding plaintexts. For instance, given a plaintext message m, the ciphertext c computed as $c \longleftarrow \mathcal{E}_{\mathsf{k}}(m)$ can be changed to c' such that both c and c' decrypt to m. Clearly, the CBC and CTR modes do not belong to such a class of encryption schemes. We stress that the results of [AB01, BN08] do not mean that all possible EwR or MtE schemes are inherently broken. However, it does mean that the security status of any instantiation of EwR or MtE is not known a priori and must be proved. Some of them are indeed broken and others are not. To illustrate, a body of results (e.g. [NRS14]) has proved the security of several schemes following the *not generically secure* MtE construction.

In this chapter, we give the first proof that for both CBC and CTR modes, HtE does not guarantee integrity regardless of the used hash function. In addition, the proof that we provide is not a mere existential forgery or a theoretical distinguishing attack. Unlike related work, we provide a practical attack that could be exploited to compromise the Android KeyStore. The threat is concrete: the broken HtE in CBC mode (Hash-then-CBC-Encrypt) is the cryptographic scheme that is used to safeguard the stored keys in Android mobile devices.

4.5 Hash-then-Encrypt Security Results

In this section, we start by reviewing the different concepts of integrity which our proof relies on. We then provide a formal definition of the composite construction Hash-then-Encrypt. We end by proving that this construction is not secure when using the encryption modes CBC and CTR.

4.5.1 Integrity Notions

Throughout this chapter, we only consider two notions of integrity: integrity of ciphertext (INT-CTXT) and ciphertext unforgeability (CUF-CPA). Please refer to the section 3.2.5 if you are not familiar with these two notions of integrity. Recall that both notions require that no adversary be able to produce a valid ciphertext which the encryption oracle had never produced before. However, contrary to INT-CTXT, the adversary in CUF-CPA has no access to the decryption oracle and outputs only one attempted forgery. Despite of their similarity, these two notions are defined to accomplish different goals. Indeed, INT-CTXT is a strong measure for security, while CUF-CPA is a strong one for the effectiveness of the potential attacks. Thus, proving that a symmetric scheme does not achieve neither INT-CTXT nor CUF-CPA entails two consequences: (1) the scheme does not provide high security and therefore it should not be used by scheme designers; and (2) the found attack is very damaging due to its readily implementation in practice.

As a matter of fact, we do not need to provide two different proofs such that a given AE scheme is not INT-CTXT and not CUF-CPA secure. Indeed, following the Result 3.3, if a scheme is not INT-CTXT, then consequently, it is not CUF-CPA. Hence, it is enough to only prove that the scheme is not INT-CTXT secure.

Nevertheless, our goal here is to explicitly provide a selective forgery upon which our attack scenario against the KeyStore is built.

4.5.2 Hash-then-Encrypt

Conceptually, Hash-then-Encrypt (HtE) in its general setting is an authenticated encryption scheme obtained from the association of any given hash function with any given encryption algorithm. A formal definition is given below.

Construction 4.1. [*Hash-then-Encrypt (HtE)*]
Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. Let h be a hash function. Without loss of generality, we suppose that $\mathcal{SE}$ is based on a block cipher E of block size l, and that h outputs a string of length l bits (otherwise some padding is needed). Given a plaintext message $m \in \{0,1\}^{nl}$, where $n \in \mathbb{N}$, we define the composite AE scheme Hash-then-Encrypt $\text{HtE}[h, \mathcal{SE}] = (\mathcal{K}\text{-HtE}, \mathcal{E}\text{-HtE}, \mathcal{D}\text{-HtE})$ as follows.

HtE Encryption $\mathcal{E}_{\mathsf{k}}$-HtE(m)

1: $\sigma \longleftarrow h(m)$
2: $c \longleftarrow \mathcal{E}_{\mathsf{k}}(\sigma || M)$
3: **return** c

HtE Decryption $\mathcal{D}_{\mathsf{k}}$-HtE(c)

1: Parse $\mathcal{D}_{\mathsf{k}}(c)$ as $\sigma' || m$
2: **if** $\sigma' \neq h(m)$ **then**
3: **return** $\perp$
4: **end if**
5: **return** m

4.5.3 Hash-then-Encrypt is IND-CPA Secure

Here, we prove that the composite scheme inherits the IND-CPA security goal of its underlying encryption scheme regardless of the hash function.

Theorem 4.1. [*HtE is IND-CPA*]
Let h be a hash function and let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an IND-CPA symmetric encryption scheme. Let $\mathrm{HtE}[h, \mathcal{SE}] = (\mathcal{K}\text{-HtE}, \mathcal{E}\text{-HtE}, \mathcal{D}\text{-HtE})$ be the associated AE scheme formed by following the HtE construction. Consider any IND-CPA adversary $\mathcal{A}$ attacking $\mathrm{HtE}[h, \mathcal{SE}]$, there is a negligible function negl such that:

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{A},\,\mathrm{HtE}}(k) \leq \mathsf{negl}(k)$$

Proof. We construct our proof using the principle of reduction (see section 3.2.1). We start by defining $\mathcal{B}$: an IND-CPA adversary against $\mathcal{SE}$ that we associate to $\mathcal{A}$. Recall that the goal of $\mathcal{B}$ is to simulate the execution environment of $\mathcal{A}$ who makes queries of the form (m_0, m_1) and expects answers from the oracle $\mathcal{E}_{\mathsf{k}}\text{-HtE}(\mathcal{LR}(.,.,b))$. The algorithm of $\mathcal{B}$ is described below:

Algorithm $\mathcal{B}^{\mathcal{E}_{\mathsf{k}}(\mathcal{LR}(.,.,b))}$

```
repeat
    if A queries (m_0, m_1) then
        m'_0 ⟵ h(m_0)||m_0; m'_1 ⟵ h(m_1)||m_1
        c ⟵ E_k(LR(m'_0, m'_1, b))
        output c to A
    end if
until A outputs b'
return b'
```

We can see that $\mathcal{B}$ perfectly simulates the answers to $\mathcal{A}$. In addition, we can see that $\mathcal{B}$ returns the same output as $\mathcal{A}$ does, and thus if $\mathcal{A}$ correctly guesses b, so does $\mathcal{B}$. Hence, we have:

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{A},\,\mathrm{HtE}}(k) = \mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{B},\,\mathcal{SE}}(k)$$

Moreover, recall that the encryption scheme $\mathcal{SE}$ is IND-CPA secure. Therefore, there exists a negligible function negl such that:

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{B},\,\mathcal{SE}}(k) \leq \mathsf{negl}(k)$$

We conclude our proof by combining the two relations above. Hence, HtE is IND-CPA secure for any given hash function if the underlying encryption scheme is also IND-CPA secure. □

Now, we study the integrity of HtE. We will distinguish two cases: CBC mode and CTR mode.

4.5.4 Hash-then-CBC-Encrypt

Let us start with the case where HtE encrypts messages using the CBC mode. Henceforth, We call this case Hash-then-CBC-Encrypt (HtCBC). Here, we formally prove that HtCBC is neither INT-CTXT nor CUF-CPA secure.

Hash-then-CBC-Encrypt is not INT-CTXT

Here, we prove that HtCBC is not INT-CTXT secure. For this, we use the relations among notions that are defined in [BN08]. In particular, we use a derived one: if an AE scheme is IND-CPA secure and not IND-CCA secure, then it is not INT-CTXT (**IND-CPA** $\wedge$ **¬IND-CCA** $\Rightarrow$ **¬INT-CTXT**), which is easily obtained from the relation that was briefly presented in Result 3.2. Therefore, our proof is composed of two parts: firstly we prove that HtCBC is IND-CPA secure (already done in theorem 4.1) and secondly we prove that it is not IND-CCA secure.

Proposition 4.1. [*HtCBC is not IND-CCA Secure*]
Let h be a hash function and let $\mathrm{CBC}[E] = (\mathcal{K}\text{-CBC}, \mathcal{E}\text{-CBC}, \mathcal{D}\text{-CBC})$ be a stateless CBC encryption scheme (as it is defined in Definition 3.2), where E is a block cipher of block size l. Consider the associated AE scheme $\mathrm{HtCBC}[h, \mathrm{CBC}] = (\mathcal{K}\text{-HtCBC}, \mathcal{E}\text{-HtCBC}, \mathcal{D}\text{-HtCBC})$ formed by following the HtE construction. Then, we can construct a polynomial-time IND-CCA adversary $\mathcal{A}$ such that:

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{A},\,\mathrm{HtCBC}}(k) = 1$$

Proof. Recall that $\mathcal{A}$ has access to both the encryption oracle $\mathcal{E}_{\mathsf{k}}\text{-HtCBC}(\mathcal{LR}(.,.,b))$ and the decryption oracle $\mathcal{D}_{\mathsf{k}}\text{-HtCBC}(.)$. Also, recall that the block cipher E associated to the CBC encryption scheme is of block size l, and the hash function h outputs l bits. Therefore, no padding is needed in our construction. For $n \in \mathbb{N}$, we define the algorithm of $\mathcal{A}$ as below.

Algorithm $\mathcal{A}^{\mathcal{E}_{\mathsf{k}}\text{-HtCBC}(\mathcal{LR}(.,.,b)),\,\mathcal{D}_{\mathsf{k}}(.)}$

Let m_0 and m_1 be two messages in $\{0,1\}^{ln}$
$m'_0 \longleftarrow h(m_0)\,||\,m_0$
$m'_1 \longleftarrow h(m_0)\,||\,m_1$
$c \longleftarrow \mathcal{E}_{\mathsf{k}}\text{-HtCBC}(\mathcal{LR}(m'_0, m'_1, b))$
Parse c as $c_0||c_1||c_2||c_3$
$c' \longleftarrow c_1||c_2||c_3$
$x \longleftarrow \mathcal{D}_{\mathsf{k}}\text{-HtCBC}(c')$
if x $\neq \perp$ **then**
    return 0
else
    return 1
end if

We claim that the previous adversary succeeds whether $b = 0$ or $b = 1$. Therefore, HtCBC is not IND-CCA secure, since $\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{A},\,\mathrm{HtCBC}}(k) = 1$. Recall that the oracle $\mathcal{E}_{\mathsf{k}}\text{-HtCBC}(\mathcal{LR}(.,.,b))$ returns the ciphertext of one of the two submitted messages. Thus, we have $c = \mathcal{E}_{\mathsf{k}}(m'_b = h(m_0)||m_b)$. Applying the encryption of HtCBC, c can be written as $\mathcal{E}_{\mathsf{k}}\text{-CBC}(h(h(m_0)||m_b)\,||\,h(m_0)\,||\,m_b)$, which is composed as follows:

$$c = c_0 \;||\; \overbrace{E_{\mathsf{k}}(c_0 \oplus h(h(m_0)||m_b))}^{c_1} \;||\; \overbrace{E_{\mathsf{k}}(c_1 \oplus h(m_0))}^{c_2} \;||\; \overbrace{E_{\mathsf{k}}(c_2 \oplus m_b)}^{c_3}$$

We see that for c', c_0 is removed and c_1 becomes the new initial value. Considering the new IV, the CBC decryption algorithm performed over c' returns the rest of the plaintext $h(m_0)||m_b)$. Therefore, $\mathcal{D}_{\mathsf{k}}\text{-HtCBC}(c')$ outputs m_0 when $b = 0$, $\perp$ otherwise (unless $h(m_0) = h(m_1)$), which concludes our proof. □

Hash-then-CBC-Encrypt is not CUF-CPA

Here, we provide a selective forgery against HtCBC. Indeed, we construct a CUF-CPA adversary who succeeds in forging a valid ciphertext for any message m of her choice without requiring the secret key used for encryption/decryption. This forgery is important for the rest of the chapter, since we will use the presented selective forgery later to compromise the security of the KeyStore. Therefore, we require that the forgery be practical and not demanding in term of resources. We argue that our forgery satisfies these conditions of feasibility: the adversary succeeds after only one query to the encryption oracle.

Theorem 4.2. [*HtCBC is not CUF-CPA Secure*]
Let h be a hash function and let $\mathrm{CBC}[E] = (\mathcal{K}\text{-CBC}, \mathcal{E}\text{-CBC}, \mathcal{D}\text{-CBC})$ be a stateless CBC encryption scheme (as it is defined in Definition 3.2), where E is a block cipher of block size l. Consider the associated AE scheme $\mathrm{HtCBC}[h, \mathrm{CBC}] = (\mathcal{K}\text{-HtCBC}, \mathcal{E}\text{-HtCBC}, \mathcal{D}\text{-HtCBC})$ formed by following the HtE construction. Then, we can construct a polynomial-time CUF-CPA adversary $\mathcal{A}$ such that:

$$\mathbf{Adv}^{\text{suf-cpa}}_{\mathcal{A},\,\mathrm{HtCBC}}(k) = 1$$

Proof. Recall that $\mathcal{A}$ has access to the encryption oracle $\mathcal{E}_{\mathsf{k}}\text{-HtCBC}(\mathcal{LR}(.,.,b))$ and only to one attempt to the decryption oracle $\mathcal{D}_{\mathsf{k}}\text{-HtCBC}(.)$. Also, recall that the block cipher E associated to the CBC encryption scheme is of block size l, and the hash function h outputs l bits. Therefore, no padding is needed in our construction. We show that $\mathcal{A}$ is able to produce a valid ciphertext for any message m. For $m \in \{0,1\}^{ln}$, where $n \in \mathbb{N}$, we define the algorithm of $\mathcal{A}$ as below.

Algorithm $\mathcal{A}^{\mathcal{E}_{\mathsf{k}}\text{-HtCBC}}(m)$

1: $m' \longleftarrow h(m)||m$
2: $c \longleftarrow \mathcal{E}_{\mathsf{k}}\text{-HtCBC}(m')$
3: Parse c as $c_0||c_1||c_2||...||c_{n+2}$
4: $c' \longleftarrow c_1||c_2||...||c_{n+2}$
5: **return** c'

As mentioned in definition 3.12, the adversary A wins if the output ciphertext c' is both new and valid. Trivially, c' has never been produced by the encryption oracle $\mathcal{E}_{\mathsf{k}}\text{-HtCBC}(.)$ before, and thus it is new. In addition, we argue that the oracle $\mathcal{D}_{\mathsf{k}}\text{-HtCBC}(.)$ on c' will not return $\perp$. Indeed, using the same arguments given in proposition 4.1, c' could be written as $\mathcal{E}_{\mathsf{k}}\text{-CBC}(h(m)||m)$. Thus, $\mathcal{D}_{\mathsf{k}}\text{-HtCBC}(c') = m(\neq \perp)$. □

4.5.5 Hash-then-CTR-Encrypt

Now, we consider the case when HtE encrypts data with the CTR mode. Henceforth, we call this Hash-then-CTR-Encrypt (HtCTR). Here, we only provide the proof such that HtCTR is not INT-CTXT secure (and hence it is not CUF-CPA secure).

Hash-then-CTR-Encrypt is not INT-CTXT

Using the aforementioned arguments presented in the case of HtCBC in section 4.5.4, we show that HtCTR is not INT-CTXT secure by proving that it achieves IND-CPA but not IND-CCA. As a matter of fact, the theorem 4.1 demonstrates that HtCTR is indeed IND-CPA secure, since the given proof is independent of the used encryption mode, and thus remains true for HtCTR. Therefore, we only provide the proof that the construction HtCTR is not INT-CCA secure.

Proposition 4.2. [*HtCTR is not IND-CCA Secure*]
Let h be a hash function and let $\mathrm{CTR}[F] = (\mathcal{K}\text{-CTR}, \mathcal{E}\text{-CTR}, \mathcal{D}\text{-CTR})$ be a CTR encryption scheme (as it is defined in Definition 3.3), where F_{k} is a PRF on $\{0,1\}^l$. Consider the associated AE scheme $\mathrm{HtCTR}[h, \mathrm{CTR}] = (\mathcal{K}\text{-HtCTR}, \mathcal{E}\text{-HtCTR}, \mathcal{D}\text{-HtCTR})$ formed by following the HtE construction. Then, we can construct a polynomial-time IND-CCA adversary $\mathcal{A}$ such that:

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{A},\,\mathrm{HtCTR}}(k) = 1$$

Proof. Recall that $\mathcal{A}$ has access to both the encryption oracle $\mathcal{E}_{\mathsf{k}}\text{-HtCTR}(\mathcal{LR}(.,.,b))$ and the decryption oracle $\mathcal{D}_{\mathsf{k}}\text{-HtCTR}(.)$. We define the algorithm of $\mathcal{A}$ as below.

Algorithm $\mathcal{A}^{\mathcal{E}_{\mathsf{k}}\text{-HtCTR}(\mathcal{LR}(.,.,b)),\,\mathcal{D}_{\mathsf{k}}\text{-HtCTR}}$

```
Let m_0, m_1 and m_2 be three messages in {0,1}^l
c ⟵ E_k-HtCTR(LR(m_0, m_1, b))
Parse c as c_0||c_1||c_2
c'_1 ⟵ h(m_2) ⊕ h(m_0) ⊕ c_1
c'_2 ⟵ m_2 ⊕ m_0 ⊕ c_2
c' ⟵ c_0||c'_1||c'_2
x ⟵ D_k-HtCTR(c')
if x ≠ ⊥ then
    return 0
else
    return 1
end if
```

We claim that the previous adversary succeeds whether $b = 0$ or $b = 1$, hence its advantage is equal to one and as a result HtCTR is not IND-CCA secure. It is easy to see that $c = \mathcal{E}_{\mathsf{k}}\text{-CTR}(h(m_b)||m_b)$. Consequently, c can be written as follows:

$$c = c_0 \;||\; \overbrace{h(m_b) \oplus F_{\mathsf{k}}(c_0+1)}^{c_1} \;||\; \overbrace{m_b \oplus F_{\mathsf{k}}(c_0+2)}^{c_2}$$

Since the definition of c' includes c_1 and c_2, c' can be expanded thusly:

$$c' = c_0 \;||\; \overbrace{h(m_2) \oplus h(m_0) \oplus h(m_b) \oplus F_{\mathsf{k}}(c_0+1)}^{c'_1} \;||\; \overbrace{m_2 \oplus m_0 \oplus m_b \oplus F_{\mathsf{k}}(c_0+2)}^{c'_2}$$

Let us first take with the case when $b = 0$. c' becomes $c_0 \,||\, h(m_2) \oplus F_{\mathsf{k}}(c_0+1) \,||\, m_2 \oplus F_{\mathsf{k}}(c_0+2)$. Therefore, the following relation holds: $c' = \mathcal{E}_{\mathsf{k}}\text{-CTR}(h(m_2)||m_2)$, which makes $\mathcal{D}_{\mathsf{k}}\text{-HtCTR}(c') = m_2$ ($\neq \perp$). Now, we consider the case when $b = 1$. We can easily see that $\mathcal{D}_{\mathsf{k}}\text{-HtCTR}(c') = \perp$, unless we have $h(m_0 \oplus m_1 \oplus m_2) = h(m_0) \oplus h(m_1) \oplus h(m_2)$. This relation is unlikely to hold for a hash function (except for homomorphic hash functions [KFM04]), which concludes our proof. □

4.6 The Android KeyStore

As mentioned in section 2.3.4, the Android KeyStore is a high-level service that enables applications to store their credentials. The KeyStore is comprised of three layers: *Public APIs*, *Keystore service*, and *Keymaster*. The security of keys is primarily ensured by the *Keymaster* which is designed to protect keys from extraction. This implies that it is the only component that has a direct access to keys material, and therefore keys are represented differently outside Keymaster: alias (name) in *Public APIs* and key handlers in *Keystore service*. It is worth mentioning that hereafter all the implementation details that we provide concern the KeyStore of the build `android-6.0.1_r22`.

Generally speaking, the key handler is an opaque object that identifies a keymaster-protected key. Key handlers are implementation-dependent. We only consider the default software-only keymaster provided by Google. By inspecting its implementation found in `keymaster_openssl.cpp`, we see that the key handler is just an encoded version of the corresponding key. Encoding is achieved by concatenating a header of describing meta data to the key. The header includes: a 4-byte constant value for software keys, a 4-byte key type, and a 4-byte big endian integer for key length. Thus, the default key handler is written as follows:

`Soft_Key_Magic` || `Key_Type` || `Key_Length` || `Key`.

Our target in this chapter is the stored keys on mobile device. Therefore, in what follows, we focus solely on the secure mechanism performed by the *Keystore service* for storing keys (or more precisely key handlers).

4.6.1 KeyStore Service

Similar to other Android services, the Keystore service spans two layers in the Android architecture: the Java world (application framework) and the native world (system service). Based on the Binder design, its different components, `KeyStore.java` and `Keystore.cpp`, perform their IPC via the Binder proxy `IKeyStoreService`.

The implementation [And16d] of the Keystore reveals how the blobs of key handlers are stored on mobile device. A key handler blob (binary large object) contains a serialized version of the key handler. The keystore saves its files in `/data/misc/keystore`, where there is one directory for each user. Each directory includes files that have the following content:

- A single master key. The Keystore service is initialized by generating a 128-bit master key using the internal entropy source `/dev/urandom`. The master key is then encrypted by a 128-bit AES key derived from the screen passcode by applying PBKDF2 [Kal00]. The derivation process employs 8192 iterations with a randomly generated 128-bit salt. The encrypted keymaster, together with the salt, are stored in the `.masterkey` file.

- Key handler blobs related to user's applications. Each file contains a header of meta data as well as the encryption of the key handler using Hash-then-CBC-Encrypt. The content of the file is written as follows:

$$\texttt{meta data} \,||\, \mathcal{E}_{master\_key}\text{-CBC[AES]}(\texttt{MD5(key handler)} \,||\, \texttt{key handler})$$

We note that the KeyStore applies the AE encryption scheme $\text{HtCBC}[\texttt{MD5}, \text{CBC}[AES]] = (\mathcal{K}\text{-HtCBC}, \mathcal{E}\text{-HtCBC}, \mathcal{D}\text{-HtCBC})$ to protect key handlers. Therefore, the adversary defined in section 4.5.4 is able to maliciously forge new key handlers given valid ones. However, this attack fails in practice when performed against the real-world implementation of Keystore service because the produced key handlers would yield errors while being decoded. We recall that key handlers have a special encoding format that is specified by the keymaster. In the sequel, we adapt our forgery attack so that an adversary could fabricate a valid key handler which the keymaster successfully parses to its related key.

4.6.2 The Keys Formats

In order to conduct our attack, among all other operations provided by the KeyStore, only those involving the encryption of the stored keys will be relevant to us. This includes two operations: *key generation* and *key import*.

The KeyStore is designed to work not only with its own keys, but with those generated by a third party system. This implies that all keys, generated or imported, must follow a special format when being serialized. For instance, the keymaster requires formatting keys before wrapping them inside key handlers. The file `keymaster_defs.h` shows that there are three categories of formats:

```
typedef enum {
     KM_KEY_FORMAT_X509 = 0, /* for public key export */
     KM_KEY_FORMAT_PKCS8 = 1, /* for asymmetric key pair import */
     KM_KEY_FORMAT_RAW  = 3, /* for symmetric key import */
} keymaster_key_format_t;
```

We notice that standard formats (i.e. X.509 and PKCS#8) are used for key-pairs (e.g. RSA, DSA and EC), while no format is provided for symmetric keys (e.g. HMAC). Thus, the exact bytes comprising a symmetric key are encapsulated inside the stored key handler. This is due to the fact that the support for symmetric keys is quite recent (since Android 5.0).

This lack of formatting makes the adversary task easier. Indeed, it is hard to fabricate a ciphertext that is both valid and properly formatted. Consequently, the current version of our attack is limited to applications using symmetric keys.

4.7 Attacking the Android KeyStore

4.7.1 Threat Model

The adversary's goal is to undetectably undermine the security of the applications relying on symmetric keys for their security. For this purpose, we assume that the adversary installs some malware on the mobile device. This malware is capable of importing keys inside the KeyStore, since any installed application does have this capability. In addition, the malware is supposed to be granted the read-write permission on the KeyStore directory (i.e. `/data/misc/keystore`).

Furthermore, the malware is executed inside a mobile device with protective tools. First, the mobile system detects any malware trying to connect to a remote server. Second, the mobile system imposes the use of a strong screen passcode. This helps to avoid exhaustive attacks, since the master key of the KeyStore is derived from this passcode. Third, the system prohibits the KeyStore from storing short or obviously non-random keys. Thus, the adversary cannot perform the trivial attack consisting of generating the same key for all applications or generating a different key for each application and communicating it to a server. In both cases, the attack would be detected. We insist that these assumptions are highly plausible in corporate environments where companies enforce the security of their employees mobile devices.

Finally, the adversary controls all communications with the mobile, and thus can intercept and tamper with any exchanged message. Besides, it is assumed that any proved cryptographic mechanism is secure unless weak keys are used.

To sum up, in order to succeed her attack, the adversary should silently "break into" the KeyStore to shorten, and hence weaken, the stored keys which the targeted applications would blindly continue using.

4.7.2 The Forgery Attack

The purpose of the forgery attack is that given a ciphertext of a symmetric key, the adversary can fabricate another ciphertext that decrypts to a shorter key. As already stated, the KeyStore protects keys by encrypting their key handlers with HtCBC. Thus, keys protection, involving their confidentiality and integrity, is done using a special variant of HtCBC which we call *encode-then*-HtCBC (eHtCBC). In what follows, we adapt the selective forgery defined against HtCBC.

Loosely speaking, eHtCBC is an AE encryption scheme where messages are *encoded* before HtCBC-encrypting them. Here, encoding is performed by only appending the length of a message. To be more precise, consider a composite scheme HtCBC $= (\mathcal{K}\text{-HtCBC}, \mathcal{E}\text{-HtCBC}, \mathcal{D}\text{-HtCBC})$. Then, we define its encoded version eHtCBC $= (\mathcal{K}\text{-}e\text{HtCBC}, \mathcal{E}\text{-}e\text{HtCBC}, \mathcal{D}\text{-}e\text{HtCBC})$, where for all message m, the next relation holds:

$$\mathcal{E}_{\mathsf{k}}\text{-}e\text{HtCBC}(m) = \mathcal{E}_{\mathsf{k}}\text{-HtCBC}(\texttt{Len}(m) \,||\, m),$$

where $\texttt{Len}(.)$ is a function that takes a message as input and returns the number of blocks composing it. In what follows, we adapt the selective forgery defined against HtCBC in order to compromise the integrity of eHtCBC.

Let eHtCBC be an AE composite scheme as described above and let l be the block size of its underlying block cipher. Here, $l = 16$ bytes, since the underlying block cipher is AES. Let m be an arbitrary weak symmetric key. We define the adversary $\mathcal{A}$ who can import keys of her choice to the KeyStore, namely $\mathcal{A}$ has access to the `import` oracle. For the sake of clarity, we omit the constant values in the header of the key handler, and so only the `key_length` is kept. Therefore, the `import` function corresponds to the encryption algorithm related to eHtCBC (i.e., $\texttt{import}(m) = \mathcal{E}_k\text{-}e\text{HtCBC}(m)$). It is worth mentioning that this simplifying assumption does not alter the logic of the attack. The adversary $\mathcal{A}$ wins if she can produce a valid eHtCBC-ciphertext of m. However, conforming to our threat model (section 4.7.1), she cannot import m directly. To this end, $\mathcal{A}$ executes the algorithm below:

Algorithm $\mathcal{A}^{\texttt{import}(.)}(m)$

1: $m' \longleftarrow \texttt{Len}(m) \,||\, m$
2: $m'' \longleftarrow \texttt{padding} \,||\, \texttt{MD5}(m') \,||\, m'$
so that $\texttt{Len}(.)||\texttt{padding}$ is an l-block
3: $c \longleftarrow \texttt{import}(m'')$
4: Parse c as $c_0||c_1||c_2||c_3||c'$
5: $c'' \longleftarrow c_2 \,||\, c_3 \,||\, c'$
6: **return** c''

Two points should be noted in the described algorithm. We first give a closer look at the structure of m''. We can see that $\texttt{Len}(m'') = \texttt{Len}(m) + 2$, since $\texttt{MD5}(.)$ outputs a string of length l $(= 16)$ bytes, and `padding` is just some dummy data so that $\texttt{padding}||\texttt{Len}(.)$ constitutes one l-block. Second, c is indeed computed as $\mathcal{E}_\mathsf{k}\text{-HtCBC}(\texttt{Len}(m'') \,||\, m'')$. Thus, using the constructions of HtCBC and its encoded version eHtCBC, we see that c is composed as follows (where E is the block cipher AES and $\mathcal{E}_\mathsf{k}\text{-ivCBC}(.,.)$ is as described in Definition 3.1):

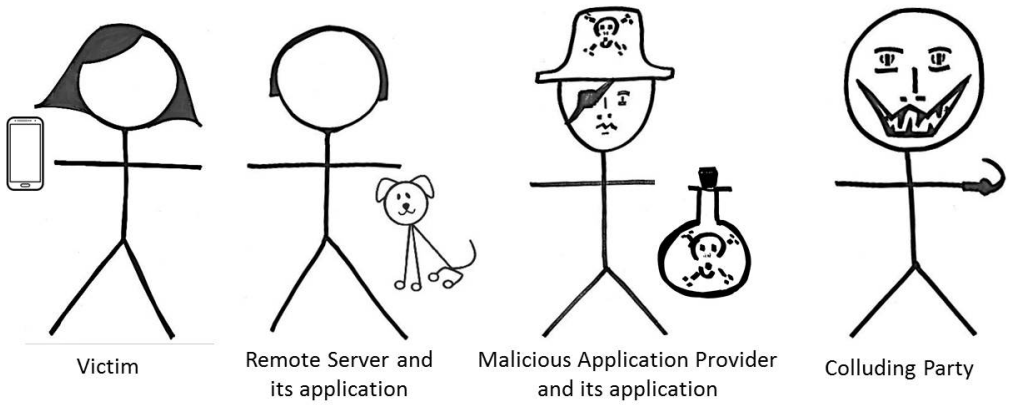
$$c = c_0 \,||\, \overbrace{E_\mathsf{k}(c_0 \oplus \texttt{MD5}(\texttt{Len}(m'')||m''))}^{c_1} \,||\, \overbrace{E_\mathsf{k}(c_1 \oplus \texttt{Len}(m'')||\texttt{padding})}^{c_2}$$
$$||\, \overbrace{E_\mathsf{k}(c_2 \oplus \texttt{MD5}(\texttt{Len}(m)||m))}^{c_3} \,||\, \overbrace{\mathcal{E}_\mathsf{k}\text{-ivCBC}(c_3, \texttt{Len}(m)||m)}^{c'}$$

Therefore, following the same arguments provided in the proposition 4.1 and the theorem 4.2, we deduce that $\mathcal{D}_\mathsf{k}\text{-}e\text{HtCBC}(c'')$ outputs m, which means that $\mathcal{A}$ achieves its goal. Though, it is important to notice that the adversary owes part of her success to the absence of verification of sound key lengths. Indeed, considering all the technical details that we provided, the length in bytes of the actual imported key m'' is always greater than 32, since it is constructed of at least two AES blocks ($\texttt{Len}(m'') = \texttt{Len}(m) + 2$). For instance, if the adversary $\mathcal{A}$ selects 4-byte m (or key), it calls the `import` function on a key of length 36 bytes. We recall that AES keys cannot be longer than 32 bytes. Fortunately (for the adversary), no checking is done by `import`, and consequently the attack ends successfully.

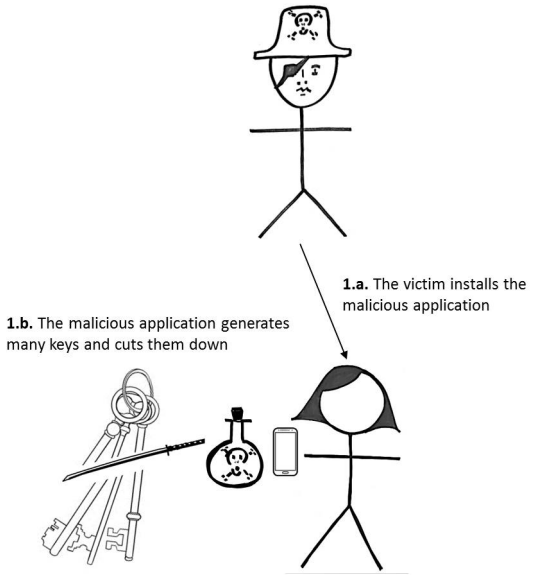
We underline that the interest of the above attack is twofold. First, it can be abused by some malware to breach the KeyStore security even in a well-protected mobile system. Second, we prove that encoding does not improve the security of HtCBC unlike for many other AE schemes. We believe that this result is of independent importance regardless of the introduced attack scenario.

4.7.3 The Undetected Malware

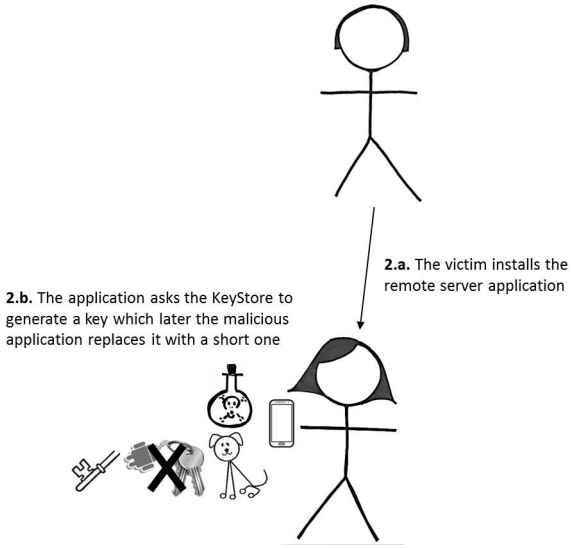
We illustrate the fallout of our forgery against the KeyStore by a complete attack scenario. We emphasize that the severity of protecting highly sensitive data, like keys, by a broken crypto-



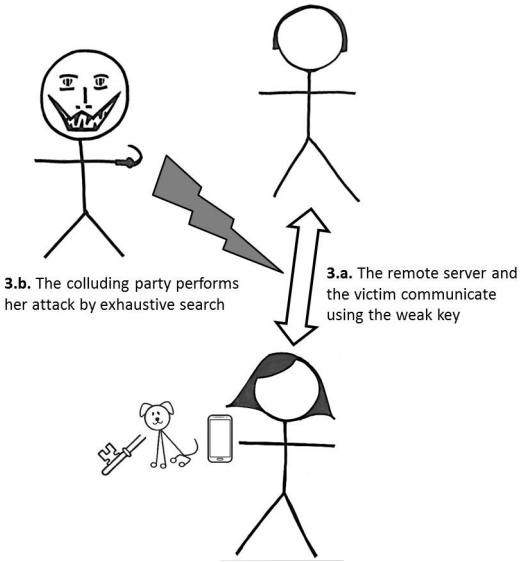
(a) The Attack Actors



(b) The Provisioning Phase



(c) The Lulling Phase



(d) The Attacking Phase

Figure 4.1: Overview of the Forgery Attack Against the Android KeyStore

graphic scheme is not limited to the suggested scenario.

In our scenario, illustrated in Figure 4.1, the intent of the attacker is to maliciously modify all the exchanged messages between an app and a remote server even if they are protected by proved cryptography. This is possible thanks to some malware installed on the mobile and which soundlessly weakens the keys of the KeyStore. This attacker represents a new kind of threat, since she can go undetected while compromising the security of users including those hiding behind secure protocols.

Actors

We define five actors to describe the plot of the attack:

1. *Security Manager* who enforces the security of the mobile system. In particular, the KeyStore refuses to store weak (i.e. short) keys. Additionally, the system would detect any malware trying to communicate with its accomplice server;

2. *Victim* who uses the said mobile to perform some services requiring to protect their critical transactions. The corresponding cryptographic keys are managed by the KeyStore;

3. *Remote Server* related to the running services and to which the sensitive data are sent. It is important to note that this server does not possess the required keys to decrypt the sensitive data. Its role is mainly to keep an encrypted copy of users data, thereby allowing them to recover their data later. Despite being naive, such principle is not only used for *sealed storage*, but also in numerous protocols for various purposes such as authentication in FIDO U2F [SBTC15]; the new standard of two factor authentication for web servers.

4. *Malicious Application* viciously shortening the keys of other applications;

5. *Colluding Party* that is able to intercept and alter any exchanged message on the network.

Attack Workflow

We suppose that the attacker has already convinced the victim in some way to install the malicious application on her device. The attack scenario is structured into three phases: provisioning, lulling and attacking.

Provisioning phase. The malicious application runs in background and executes the algorithm described in section 4.7.2. Thus, it craftily generates several symmetric keys of length $32 + x$ bytes. Then, it imports these keys into the KeyStore which accepts them for two reasons: they are seemingly strong and no verification is done concerning their abnormal length. Afterward, it cuts them down into keys of length x bytes. Here, we take x to be 4, so that keys are small enough to allow a swift brute-force attack. For the sake of completeness, we precise that once the keys are trimmed, their meta data are required to be padded with some dummy data. This is because the files containing the keys must remain of constant size. For brevity, we omit the technical details related to this balancing operation.

Lulling phase. In this phase, an application on the victim's device asks the KeyStore to generate a key with *alias* as its name. The malicious application, snooping on the KeyStore, notes this alias as well as the UID of the caller application. As soon as the key is generated and its associated file is created, the malicious application modifies the name of one of its keys in such a way that the renamed key is believed to belong to the targeted application. Some might argue that this operation is delicate: the malicious application is assumed to continuously supervise the KeyStore activities. Nevertheless, we argue that no special privilege is required to do this, since it can be

done with quite ease by regularly monitoring the content of the KeyStore folder. Indeed, the key's alias and the creating application's UID could be quickly guessed from the key file name. Now, let us investigate the case when the concerned application has succeeded in encrypting some data with its generated key before being replaced by the malicious application. The risk of such case is that the victim would detect that she is actually under attack. We can see that this does not happen because the KeyStore will mark these encrypted data as invalid. Hence, the victim would be prompted to re-enter these maliciously (or accidentally) modified data. This is due to the fact that the KeyStore will attempt to decrypt such data using the new (weak) key which is different from the former key, and therefore the integrity verification would eventually fail.

Attacking phase. Now, the user is carrying out some operations that involve transmitting sensitive messages to a server. The application handling such operations needs to protect the integrity of these messages. Therefore, it asks the KeyStore to generate an HMAC tag over each message. The KeyStore returns a tag unwittingly generated with the weak key. Concatenated to their tag, the messages are then intercepted by the colluding party while being sent to the server. The latter performs an exhaustive search to find the secret key used to generate the HMAC tag. Since the search space being explored is shrunk, the brute-force search ends quite fast. The colluding party then modifies the content of some messages (e.g. the total amount of a payment transaction), and recomputes a valid tag for them before forwarding the new messages to the server. In this way, the attacker effortlessly breaks into victims who think that they are safe with primitives, HMAC for example, which are believed to be secure.

4.8 Discussion and Recommendations

An important aspect of any forgery is what it implies in practice. Here, we have demonstrated how a theoretical weakness could be exploited to undermine the security of a real-world system, namely Android. We insist that this scenario is just an example: **a wholly new class of threat could be built from our forgery attack**.

Moreover, it is worth noting that the attack of section 4.7 is conceived to be applicable only against software-only implementations of KeyStore. We admit that it does not directly impact hardware-backed implementations which exist on some mobile devices. Indeed, our scenario involves forging keys by forging key handlers, which can be easily done in software-only implementations. Nevertheless, generally speaking, hardware-backed implementations, such as those based on TEE, encrypt their keys to produce their key handlers. They use AE schemes in order to protect both the integrity and the confidentiality of keys. Therefore, keys integrity is protected by two means: the AE scheme of the Keystore service and that of the TEE. In our scenario, an attacker can still forge a valid key handler that is sent to the Keymaster (i.e., TEE). The TEE in its turn will detect the forgery when it decodes the forged key handler to its corresponding key, which means that the attack does not succeed. However, we can imagine other possible vectors of attack. For example, an attacker might perform a fuzzy attack by generating valid key handlers and send them to the TEE. A malformed key handler might allow the attacker to carry out, for instance, a stack overflow attack.

Furthermore, we notice that our attack is a variant of the algorithm substitution attack (ASA) defined in [BJK15]. In ASA, the attacker subverts the encryption algorithm to break the secrecy of the scheme while being undetectable. Our attack is slightly different. It violates the integrity of the scheme while being undetectable. It can be seen as a key substitution attack (KSA) in which users' keys are replaced with weak ones. The KSA and the ASA share the same formalization of detectability as well as the properties of statelessness and size-obliviousness.

Finally, we believe that even if some may argue that our attack is difficult to mount, there is value in identifying these types of design flaws. Corporate-issued devices or state-level malware

could easily execute the described attack in order to gain undetectable long-term access to device communications.

Recommendations. Having thus presented our main results, we are now on a position to make specific recommendations. Our findings indicate that the key countermeasure to nullify our attack is to replace the flawed AE scheme HtCBC by a secure one. We recall that any countermeasure intended to fix a deployed system must not cause intrusive changes that affect the entire architecture of this system. Fortunately, the KeyStore design is modular enough to allow modifying the scheme HtCBC without involving the rest.

The simplest solution would be to keep the *Hash-then-Encrypt* construction and use it with another encryption mode than CBC. The Counter (CTR) mode is often perceived as being advantageous to other modes. However, we have proved that the scheme *Hash-then-CTR-Encrypt* does not provide integrity either (refer to section 4.5.5 for the full proof). We could have proposed other encryption modes for Hash-then-Encrypt, however the lack of obvious attacks cannot be taken as evidence of the soundness of a scheme. Instead, it would be better to switch to proved schemes, such as Encrypt-then-MAC (refer to Result 3.7) or secure dedicated AE schemes. We admit that the choice of an AE scheme is never easy in practice. This is not due to the lack of secure AE schemes, but to the absence of a clear standard like AES for block ciphers. Fortunately, many efforts are being made to define such an AE scheme by participating at the Caesar competition [Ber15]. In December 2017, the Caesar competition is expected to announce the final winner that will be recommended by the same institution that has previously standardized famous cryptographic constructions, such as AES and SHA-3.

4.9 Conclusion

In this chapter, we have provided a selective forgery against the AE scheme Hash-then-CBC-Encrypt (HtCBC). In our forgery, the attacker is able to produce a fresh valid ciphertext for any message selected by the attacker prior to the attack. The defined attack is attractive to implement in real-world systems, since it is simple and not demanding in term of resources. Indeed, it only requires having the ciphertext of one chosen message in order to succeed with probability 1.

Then, we have shown that encoding does not protect HtCBC from our selective forgery. Stated differently, we have proved that the scheme Encode-then-HtCBC (eHtCBC) does not guarantee integrity. Here, the consequences are too serious to be ignored, since eHtCBC is the AE scheme which safeguards the users' keys in mobile devices running on Android. We illustrated this by demonstrating a practical scenario of a key-substitution attack: the attacker makes the victims to use weak keys for their critical transactions without their knowledge. In addition, compared to other attacks, the defined attack satisfies particular properties, such as version-independence and size-obliviousness, to make it well-suited for mass surveillance.

It is worth reiterating that proved cryptography is the way to go. Although this fact is well known to cryptographers, it has not yet been widely understood by system designers. A key lesson from this chapter is that cryptographers and system designers must work closely together. Bridging the perilous gap that separates these two communities will be essential for keeping future systems more secure.

Chapter 5

Cryptanalysis of GlobalPlatform Secure Channel Protocols

"There's only one thing you should know, I've put my trust in you"

– Linkin Park, In The End

Contents

5.1 Introduction **70**
5.1.1 Overview of our Attack 70
5.1.2 Context in Trusted Computing 71
5.2 Background **71**
5.2.1 Secure Card Content Management 72
5.2.2 Secure Channel Protocols 73
5.3 Related Work **73**
5.4 Secure Channel Protocol '2' **75**
5.4.1 Description 75
5.4.2 Try-and-Guess Attack 75
5.4.3 Plaintext Recovery Against Smart Cards 77
5.4.4 Discussion about Theoretical Feasibility 78
5.5 Secure Channel Protocol '3' **79**
5.5.1 Description 79
5.5.2 Security Models 81
5.6 SCP03 Security Results **85**
5.6.1 Sf-nEtTw Security Analysis 85
5.6.2 SCP03 Security Analysis 86
5.6.3 Practical Considerations of SCP03 Security Analysis 89
5.7 Security Proofs of Sf-nEtTw **90**
5.7.1 Proof of Proposition 5.1 90
5.7.2 Proof of Theorem 5.1 93
5.8 Discussion **95**

GlobalPlatform (GP) card specifications are the de facto standards for the industry of smart cards. Being highly sensitive, GP specifications were defined regarding stringent security requirements. In this chapter, we analyze the cryptographic heart of these requirements; i.e. the family

of Secure Channel Protocols (SCP). Our main results are twofold. First, we demonstrate a theoretical attack against SCP02, which is the most popular protocol in the SCP family. We discuss the scope of our attack by presenting an actual scenario in which a malicious entity can exploit it in order to recover encrypted messages. Second, we investigate the security of SCP03 that was introduced as an amendment in 2009*. We find that it provably satisfies strong notions of security. Of particular interest, we prove that SCP03 withstands algorithm substitution attacks (ASAs) defined by Bellare et al. that may lead to a secret mass surveillance. Our findings highlight the great value of the paradigm of provable security to increase the trust in standards and the enterprise of certification, since unlike extensive evaluation, it formally guarantees the absence of security vulnerabilities. The findings of this chapter were published in [ST16b] that was presented at the 3rd Conference on Security Standardisation Research (SSR 2016).*

5.1 Introduction

As mentioned previously in chapter 2 about the Secure Element (refer to section 2.4.2), secure elements are mainly based on the technology of smart cards. Nowadays, smart cards are already playing an important role in the area of information technology. Considered to be tamper resistant, they are increasingly used to provide security services [RE10]. Smart cards do not only owe their tamper resistance for their success; programmability is a key issue for the wide adoption of this technology. Indeed, programmability made it possible to load new applications or remotely personalize existing ones during the cards life cycle [Mar98]. However, this dynamicity did not come without price, as it has brought up security concerns about the novel system of content management. The absence of standards has motivated the creation of GlobalPlatform.

GlobalPlatform (GP) [Glo16c] is a cross-industry consortium that publishes specifications on how post-issuance management shall be carried out for smart cards. This includes the functionality to remotely manage cards content in a secure way. The cryptographic heart of these mechanisms is the family of *Secure Channel Protocols* (SCPs) which protect the exchanged messages. Optimized for cards, the used encryption schemes in these protocols do not follow any standardized or provably secure construction.

Since its first publication, the GP card specifications have been the subject of diverse verifications. For instance, authors in [B05] examine some aspects of these specifications and prove their soundness with the B method. Nevertheless, to the best of our knowledge, a rigorous analysis of the SCP encryption schemes has never been provided before. Our goal is thus to formally study them, and hence to validate (or invalidate) the security guaranteed by GP. The motivation of this work is to show the interest of the provable security approach to verify specifications published by an international cross-industry consortium working on critical architectures.

In this chapter, we apply the methods of provable security on GP specifications. We start by analyzing the most popular GP SCP (i.e. SCP02). Much to our surprise, we find that SCP02 is vulnerable to a well-known security flaw caused by encrypting data using CBC mode with no random initialization vector (IV). Then, we shift our analysis to the youngest member of the SCP family (i.e. SCP03). SCP03 encrypts messages using the "Encrypt-then-MAC" (EtM) method that is proved secure in [BN08]. In this chapter, we provide a stronger result: SCP03 provides much more than the standard goals of security.

5.1.1 Overview of our Attack

The basic idea behind our attack scenario against SCP02 is quite simple and can be explained as follows. The scenario starts when at least two entities (e.g. service providers) share the same trusted third party (e.g. trusted service manager) to communicate with smart cards. Using the confidentiality flaw of SCP02, a malicious entity can recover some encrypted messages belonging to another entity by performing a kind of try-and-guess attack. Indeed, it firstly intercepts a

ciphertext of another entity and then enters into a loop of guessing the underlying message and validating the guess with the trusted party. The attacker stops when a valid guess is found. At first glance, such an attack might seem impractical and hard to implement against real-world systems, since it requires several conditions to be met in order to succeed. In section 5.4, we discuss this attack in more detail and show its relevance in the context of smart cards.

5.1.2 Context in Trusted Computing

Independently of any immediate practical exploit, our findings tell us something about the *trust*. This chapter is about to analyze the security guaranteed by the SCP family standardized by GlobalPlatform. Being involved in sensitive services (e.g. payment), they have undergone rigorous verification and validation tests. Therefore, many cast no doubt about their soundness, and hence they are used in several systems achieving high assurance level in common criteria (CC) [Ora12]. Here, we approach this problem of trust in a different manner. Indeed, we use the tools of provable security to formally investigate how secure the members of the SCP family effectively are.

Our work highlight an interesting fact: security in well-established standards still does not withstand a simple cryptanalysis. We show, once again, that security by extensive verification or eminent authority is highly misleading. We emphasize that the presented attack against SCP02 is due to a well-known vulnerability in the domain of cryptography (i.e. the use of no random IV for the CBC mode) [KL15]. Thus, our result raises serious concerns about CC certification. We hope that our work will encourage further integration of provable security inside the enterprise of certification to improve the security of the certified protocols.

Nevertheless, this chapter does not only present bad news about the SCP family. Indeed, we prove that SCP03 satisfies the security model defined by Bellare et al. in [BKN02] which better models the particularity of SCP03. One main advantage of this model is that, in addition to satisfying the existing notions of confidentiality (i.e. IND-CPA) and integrity (i.e. INT-CTXT), it protects against replay and out-of-delivery attacks. More importantly, we prove that it defends against the recent threat of mass surveillance by algorithm-substitution attacks (ASAs) [BPR14]. Typically closed-source, the industry of smart cards is concerned about ASAs, since no code scrutiny is possible to assert the absence of backdoors in the implemented protocols. This could damage the confidence in smart cards. Our proof guarantees that SCP03 cannot undetectably contain hidden backdoors allowing mass surveillance as it is outlined in [BPR14].

5.2 Background

In order to better understand this chapter, readers are required to be familiar with some background information:

- Secure card content management and Secure Channel Protocols (described below);
- Tweakable block ciphers (section 3.1.3);
- Encryption schemes and modes of operation ($\mathcal{SE}$/CBC, section 3.1.4);
- Nonce-based symmetric encryption ($n\mathcal{SE}$, section 3.1.5)
- Message authentication schemes (MAC, section 3.1.7)
- Authenticated encryption schemes (AE, section 3.1.8);
- The principles of provable security and the proofs of reduction (section 3.2.1);
- Pseudorandom functions (PRF, section 3.2.2);

- Indistinguishability under CPA/CCA (IND-CPA/IND-CCA, section 3.2.4);
- Integrity of ciphertexts (INT-CTXT, section 3.2.5);
- Strong unforgeability (SUF-CMA, section 3.2.7).

5.2.1 Secure Card Content Management

Here, we briefly highlight the mains characteristics of content downloading/loading in a GlobalPlatform compliant smart cards. The content loading process allows a card issuer to add data to the persistent memory of the card. All commands of content loading must first be processed by a card component representing the issuer of the card called *Issuer Security Domain*. On receipt of a loading command, this component checks that it actually comes from an authenticated party.

However, it is an important prerequisite of multi-application smart cards that data can be securely sent from different parties. This is why GlobalPlatform defines the concept of Delegated Management that is a powerful building block towards the design of a truly multi-application smart card environment. Indeed, Delegated Management empowers the Card Issuer with a mechanism that allows them to preauthorize some trustworthy party with certain content management operation. We call such a party Trusted Service Manager (TSM). In this way, The TSM is provided with the flexibility of managing card applications.

Thus, the GlobalPlatform model works as follows. The Card Issuer, relying on the mechanism of Delegated Management, leases the ability of managing card content to one or several Trusted Service Managers. Now, any Service Provider requiring to install an application onto the card asks a TSM to manage its application. Therefore, several Service Providers can share the same TSM that uses the same delegation capability to manage all the applications. Using the terminology of GlobalPlatform, we say that the Card Issuer creates several Security Domains and gives them some management ability in order to 'leases' them. A TSM can have one or several Security Domains upon which the TSM relies to manage the applications of Service Providers. Below, we describe the concept of Security Domain.

Security Domains

A Security Domain (SD) allows its owner to control an application in a secure way. SDs support security services such as cryptographic functions and key handling. In terms of communicating with the outside world, SDs implement different Secure Channel Protocols, which will be extensively discussed in this chapter. In an earlier versions of GlobalPlatform API, applications could request access to the SD services. In the current version (namely v2.3 [Glo15a]), an SD may also request services from an application. In this way, an SD may initiate a service that allows the flow of information (for further processing) from an SD to an application.

As point out previously, the SDs are the *on-card* representatives of the Card Issuer or a TSM. Moreover, it is the SDs that allow Issuers to share control with approved partners over selected portions of their card. In a GlobalPlatform card, each application is assigned to a particular SD. For the sake of completeness, in the following paragraphs, we highlight the main features of, as well as the difference between, Issuer Security Domain and the TSM ones.

- Issuer Security Domain. The Issuer Security Domain (ISP) represents the Issuer's interests in the card and it also manages the Issuer's applications. It holds the Issuer's keys and performs all issuer related card content management functions. The ISD domain stores keys and performs cryptographic operations in order to verify any request for modifying the card content. The GlobalPlatform card specification does not specify how the ISD shall be implemented and instantiated in the card.

• Provider Security Domain. The Provider Security Domain (SD) allows a TSM to securely download, install and maintain a smart card application. This is achieved seamlessly and efficiently without compromising its security requirements or involving any third parties. This mechanism allows TSM to share some space in the card without the danger of compromising the security of either the card or other applications. It is the SD that delegates some content management capabilities to the TSM. This is convenient, since it saves the Card Issuer from additional administrative overheads associated with maintaining third party applications.

5.2.2 Secure Channel Protocols

The GlobalPlatform card specification defines secure communication protocols over those found in ISO 7816-4 [ISO13]. Basically, a secure channel session can be used for entity authentication and cryptographic protection of subsequent communication. For instance, a TSM can initiate a secure session and communicate with its corresponding SD in order to authenticate each other and then cryptographically protect any subsequent communication. All sensitive GlobalPlatform operations must be performed within a secure channel session. This means that only authorized and authenticated entities are allowed to communicate with the card.

The GP card specification defines secure channel protocols (SCPs). While the existing protocols are mainly used for card content management, they can also be used by applications for secure communication. For example, secure communication is required when a sensitive operation (e.g. personalizing the PIN code) is about to be performed.

Whenever a secure session is needed, the SCP executes three steps:

1. *initialization* that includes entities authentication and derivation of session keys;
2. *operation* in which exchanged data are protected; and
3. *termination* ending the session.

There are three secure channel protocols that use symmetric key cryptography for authentication and communication protection: SCP01 (defined in Appendix D of [Glo15a]), SCP02 (defined in Appendix E of [Glo15a]) and SCP03 (defined as Amendment in [Glo14]). Our target in this chapter is the encryption schemes employed during the second step. In follows, we provide more details about the *operation* of SCP02 and SCP03. SCP01 is not discussed because of its deprecation. It is worth mentioning that all given details on SCPs come from the GP card specification version 2.3 [Glo15a], which is the latest version at the time of writing this chapter.

5.3 Related Work

The two main research directions that our work targets is the attacks against the CBC mode and the analysis of authenticated encryption schemes. Here, we discuss the most relevant related work and argue how our work is distinguished from others.

Blockwise-Adaptive Attack

Formalized by Fouque et al. in [FJMV03], blockwise security has been firstly introduced one year before in [JMV02]. Its idea is simple: messages are not processed atomically in practice, so an adversary is able to get the ciphertext of a part of the message. Authors motivate this notion by attacking three encryption schemes proved to be secure against chosen plaintext attacks. We will focus solely on the case of CBC mode. The security proof of CBC in [BDJR97] holds only if all the calls to the underlying block cipher are independent from each other. This means that

the $(i-1)$th block of ciphertext must not be known before choosing the ith block of plaintext, otherwise independence is lost and the proof fails. We note that using a predictable IV could be seen as a special case, since the first block of ciphertext, which is the IV, is known in advance before choosing the message.

Despite its popularity, Mitchell in [Mit05b] (and more recently Rogaway in [Rog11]) concludes that the CBC mode involves so many security constraints that it would be better to abandon it for future designs. Indeed, a great number research effort has been dedicated to extend the weakness of CBC beyond theory. Some have adapted the vulnerability mentioned above in order to undermine the security of SSL3.0/TLS1.0 [Bar06, DR11]. Both attacks were motivated by the fact that the SSL3.0/TLS1.0 standard mandates the use of CBC encryption with chained (IVs); i.e. subsequent IVs are the last block of the previous ciphertexts. The attack of [DR11], called BEAST, is so efficient that migration to TLS1.1 has been recommended by IETF. Independently from blockwise security, authors in [BKN02], inspired from [Dai02], outline a vulnerability in the secure shell protocol (SSH) caused by the same reason: CBC encryption with chained IVs. It is worth noting that all the attacks of [Bar06, BKN02, Dai02, DR11] follow the same principle: some kind of plaintext recovery is possible when the attacker can both predict the next IV and control the first block of the message.

The presented attack against SCP02 follows a similar principle because constant IVs are always predictable. Despite similarity, we argue that the vulnerability presented in this chapter is caused by the design of SCP02 that is quite delicate to secure. Indeed, it instructs the use of CBC encryption together with CBC-MAC for computing integrity tag. Such combination creates two contradictory requirements for security. Indeed, it is proved that a random IV is necessary for CBC encryption [BDJR97], while CBC-MAC must use constant IV [BKR00].

Authenticated Encryption

As said in section 3.1.8, authenticated encryption (AE) is a symmetric encryption scheme that protects both data confidentiality and integrity (authenticity). Generic composition [BN08] is the most popular approach for numerous security protocols, such as SSH, TLS and IPsec. This approach is about combining confidentiality-providing encryption together with a message authentication code (MAC). Generally, three composition methods are considered: Encrypt-and-MAC, MAC-then-Encrypt and Encrypt-then-MAC.

The family of SCP protocols follows the paradigm of generic composition. On the one hand, SCP02 relies on the "Encrypt-and-MAC" (EaM) method. As pointed out in section 3.2.8, this composition method is not generically secure, but we do not consider this result in our analysis for two reasons. First, chosen ciphertext attacks are not included in our threat model, since they are hardly applicable. Indeed, the decryption operation in SCP02 is only performed by smart cards that are unlikely to misbehave (due to their tamper-resistance property and their controlled content management). Smart cards keep the decrypted messages and never output them outside, otherwise of course encryption would be of no use. Therefore, an attacker can never obtain the result of the decryption operation. Second, the used cryptographic schemes for both encryption and authentication are not secure, hence the proof of [BN08] does not hold. This means that simpler attacks exist against SCP02. We provide further details about this in section 5.4.

On the other hand, SCP03 utilizes the "Encrypt-then-MAC" (EtM) method which is proved to satisfy standard security notions: confidentiality (IND-CPA) and integrity of messages (INT-CTXT). We note that EtM is ill-suited to formalize all the power of SCP03. In this chapter, we prove that SCP03 protects against a wider range of attacks, thanks to its stateful decryption. Of particular value, we prove that it withstands replay attacks and that any secret subversion of SCP03 for malicious goals can be detected. This is an important feature, since the absence of source code might cast doubts on the trustworthiness of smart cards.

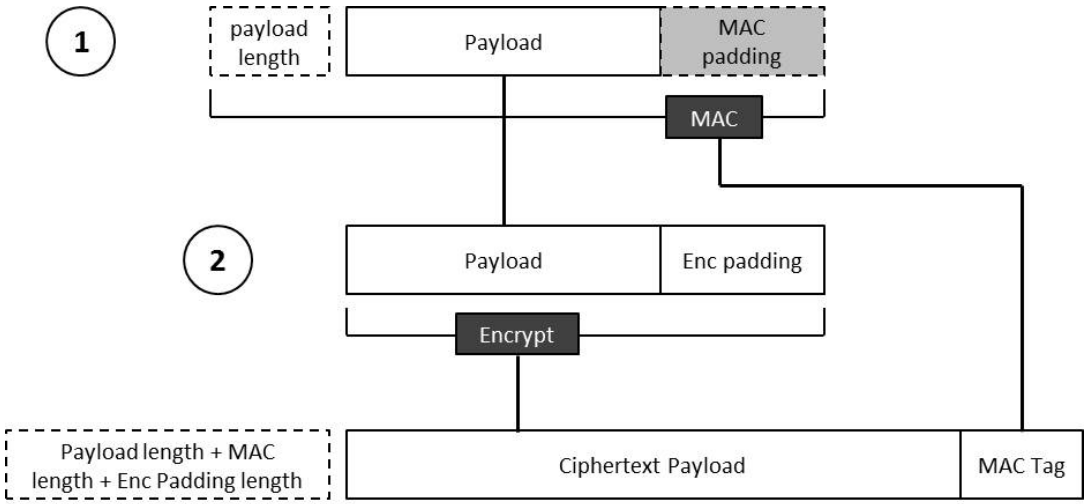


Figure 5.1: Ciphertext Generation by SCP02. Grey boxes, i.e. 'MAC padding', are not included in the 'encrypt' operation.

5.4 Secure Channel Protocol '2'

5.4.1 Description

SCP02 is the recommended protocol in the GP card specifications. It is built upon symmetric encryption based on block ciphers, hence the need of secret keys and padding data. Informally, it uses "Encrypt-and-MAC" construction, wherein the message is both encrypted and integrity protected (by using a MAC algorithm). The MAC value is appended to the encrypted message to produce the ciphertext.

In more detail, padding is first added to the message and a MAC tag is computed over the resulted data. Then, the payload is encrypted after stripping off the MAC padding to replace it by a payload one. Padding is done with binary zeroes started by $0x80$. Figure 5.1 schematically shows the ciphertext format.

Concerning the schemes in use, SCP02 mandates to encrypt data using *triple DES* in CBC mode [ISO06] with no IV, namely IV of binary zeroes (refer to Section E.4.6 in [Glo15a]). As for MAC computing, it uses a chained version of ISO9797-1 MAC algorithm 3, which includes a CBC-MAC processing with a simple DES and a Triple DES computation for the last block of the message. As a security enhancement, the last valid MAC tag is DES-encrypted before being applied to the calculation of the next MAC.

We note that both schemes are vulnerable. Indeed, authors in [FT14] perform a side-channel attack to defeat the ISO9797-1 MAC algorithm 3. Their attack allows one to recover the secret key used for the MAC computation. The consequences of such attack are limited for the reason that SCP02, like any SCP, generates the MAC tag using a temporary session key. Therefore, we do not consider this attack in the rest of the chapter. In the sequel, we describe how an attacker might exploit the absence of random IVs to recover encrypted messages.

5.4.2 Try-and-Guess Attack

Given an IV CBC encryption scheme (as defined in section 3.1), it is easy to see that for a fixed iv the CBC encryption $\mathcal{E}_{\mathsf{k}}$-ivCBC$(iv, .)$ is a stateless deterministic function of the key K for all $m \in \{0,1\}^*$. Indeed, it always yields the same ciphertext when encrypting the same message multiple times (using the same key). This has both theoretical and practical consequences.

Theoretically, it violates the security goal IND-CPA. An adversary can tell which message was encrypted after only two queries: the first query contains the same plaintext m twice, while the

second one includes m together with another plaintext. The adversary succeeds with a probability 1, since if m was encrypted, the encryption oracle would output the same result as for the first query. The equality holds because the encryption algorithm used by SCP02 is deterministic. For more clarity, we describe this attack in a more formal way.

Let E be a block cipher of l block size and let ivCBC $= (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be its associated IV CBC encryption scheme. Consider the encryption scheme $\mathcal{SE}$-SCP02 $= (\mathcal{K}\text{-SCP02}, \mathcal{E}\text{-SCP02}, \mathcal{D}\text{-SCP02})$, where $\mathcal{E}_{\mathsf{k}}\text{-SCP02}(m) = \mathcal{E}_{\mathsf{k}}\text{-ivCBC}(\mathbf{0}^l, m)$ and $\mathcal{D}_{\mathsf{k}}\text{-SCP02}(c) = \mathcal{E}_{\mathsf{k}}\text{-ivCBC}(\mathbf{0}^l, c)$ for all m and c. Then, we can construct a polynomial-time IND-CPA adversary $\mathcal{A}$ such that

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{A}, \mathcal{SE}\text{-SCP02}}(k) = 1$$

We define the algorithm of $\mathcal{A}$ as follows.

Algorithm $\mathcal{A}^{\mathcal{E}_{\mathsf{k}}\text{-SCP02}(\mathcal{LR}(.,.,b))}$

```
Let m_0 and m_1 be two messages in {0,1}^l
c ⟵ E_k-SCP02(LR(m_0, m_0, b))
c' ⟵ E_k-SCP02(LR(m_0, m_1, b))
if c ≠ c' then
    return 1
else
    return 0
end if
```

We note that the adversary succeeds due to the fact that the IND-CPA experiment (refer to Definition 3.10) does not constrain the adversary from submitting queries of the form (m, m) to the encryption oracle.

In practice, an eavesdropper observing the stream of ciphertexts is able to determine whether two ciphertexts come from the same message. Better yet, the eavesdropper can detect whether two messages share the same prefix. This could be useful to study the structure of the encrypted stream by recognizing the presence of the same data multiple times. Now, we turn the above scenario into a more serious attack.

Consider an adversary $\mathcal{A}$ who can mount a chosen-plaintext attack. $\mathcal{A}$ starts by observing a ciphertext c $(= \mathcal{E}_{\mathsf{k}}\text{-SCP02}(m))$. Recall that the goal of A is to find the message m. $\mathcal{A}$ achieves her goal by repeatedly trying all possible values for m until the correct one is identified. For instance, if the adversary knows that m is one of N possible values, then she can determine the actual value of m after $N/2$ (on average) guesses. We describe the algorithm of $\mathcal{A}$ below.

Algorithm $\mathcal{A}^{\mathcal{E}_{\mathsf{k}}\text{-SCP02}(.)}$

```
Get c from eavesdropping
found ⟵ false
repeat
    m ⟵ guess(c)
    c' ⟵ E_k-SCP02(m)
    if c = c' then
        found ⟵ true
    end if
until found = true
return m
```

where `guess` is a function that takes a ciphertext c as input and returns one possible decryption of c for each call. We notice that the adversary keeps on making guesses until finding the message

that encrypts to the eavesdropped ciphertext. Therefore, this attack is efficient against data with limited values and thus of low entropy, but it is worthless in case the exchanged data takes random values or their format is not known in advance.

We acknowledge that Try-and-Guess attack (TaGa) as outlined above has been previously suggested in other contexts (see [Bar06, Dai02, DR11]). Nevertheless, we believe that there is value in reiterating the discussion about this security flaw. The fact that the *de facto* standard of the sensitive industry of smart cards is still vulnerable to such attacks is of great interest. It indicates how the security community is divided between those designing theoretical cryptosystems and those implementing them in the real-world. We hope that our work would constitute a step towards bridging this gap.

In view of the ongoing popularity of SCP02, we believe that this vulnerability has not been identified yet. To the best of our knowledge, our work is the first one to apply TaGa in the context of GP specification for smart cards.

5.4.3 Plaintext Recovery Against Smart Cards

Here, we illustrate the fallout of TaGa by a theoretical, yet real-world attack scenario. Our attack applies to smart cards following the GP model for content management.

Actors

We define four actors to describe the plot of our attack:

1. *trusted service manager* (TSM) who owns a security domain on a smart card;
2. *victim* who uses the said smart card to execute some critical services;
3. *honest service provider* offering a sensitive service to the victim (e.g. payment); and
4. *malicious service provider* that offers some service to the victim, but mainly aims to compromise the other services.

The two service providers (the honest and the malicious) rely on the same TSM to install their applications on the victim's smart card.

Threat Model

The intent of the attacker is to recover some sensitive data related to applications installed on smart cards.

For this purpose, we assume that the adversary is capable of installing an application on the targeted smart card. Any service provider does have this ability via a TSM. In addition, the application which the adversary is supposed to install includes no harmful behaviors. In particular, it does not attempt to attack the card system. Moreover, we assume that the adversary partially controls all communications with the card: she can drop and eavesdrop any exchanged message.

Finally, we suppose that the adversary is targeting a well-protected smart card, hence no direct attack is possible. This implies several assumptions. First, the card system, properly designed, shall contain no logical security flaw. Second, the card shall implement the appropriate countermeasures to withstand hardware attacks. Third, its security domains shall be created and personalized with random keys. We emphasize that these assumptions are highly plausible for the smart card industry where products undergo extensive verification tests [Ora12].

To sum up, in order to succeed her attack, the adversary should succeed in recovering the data while being transferred between the card and the TSM. This implies to break the encryption scheme implemented by the security domain. We notice that our model allows a remote and

software-only attack. Thus, it represents a new kind of threat in the context of smart cards, since most related work involve some sophisticated hardware attacks [ABF+03, Hem04]. Our model provides several advantages over those defined in the literature (and mainly related to hardware attacks), since it concerns a large number of smart cards regardless of their manufacturer without any modification. This is because GlobalPlatform is a cross-industry consortium, and thus its specifications are adopted by numerous manufacturers. Indeed, our attack solely involves details defined in the GP card specifications which are common to all GP compliant cards.

Attack Workflow

Here, we depict how the attack scenario is performed. We suppose that the attacker has already convinced the TSM to install her application. The attack is structured into two phases.

During the first phase, the honest service provider needs to personalize her application with some secret data. She sends her query to the TSM that is responsible to carry out the secure communication to the smart card. The malicious provider detects this event and reacts accordingly. She starts by asking the TSM to send some dummy query to the smart card. Thus, the TSM shares the established secure session with the two service providers. Afterward, the attacker intercepts the encrypted messages, grabs that of the honest service provider, and drops hers (easily recognizable by, for instance, its header).

As for the second phase, the attacker makes some guess, asks the TSM to encrypt it, and then intercepts the produced ciphertext which she discards whenever the guess was wrong. The attacker repeats this until she succeeds.

One technical issue might rise by this scenario: SCP02 instructs to double-encrypt sensitive data by the TSM. Data are firstly encrypted by ECB (Electronic Code Book) mode before applying the encryption of the secure channel. We argue that this has no impact on our attack, since the overall encryption remains stateless and deterministic. Indeed, ECB is deterministic, and the composition of two deterministic functions is clearly deterministic.

5.4.4 Discussion about Theoretical Feasibility

Several conditions must be met before the attacker can successfully recover some sensitive data. Below, we present these conditions and discuss their relevance.

Using One Common Session. First and foremost, the attacker must encrypt her test cases with the same key that encrypts the data to be recovered. This supposes that the TSM shares a secure session between different service providers. Some might argue that this is not a trivial requirement, and therefore our attack scenario cannot be mounted in practice. However, we argue that session sharing is not uncommon for three reasons. First, there is no mention in the specifications that could be understood as it is *bad* practice to share sessions or even SD. Second, SCP02 generates its session keys by encrypting some constants concatenating to a 2-byte counter. Thus, the TSM must change its master key after only 2^{16} sessions, which makes the TSM very eager to optimize the opening of secure sessions. Third, being expensive, the TSM is also eager to reduce the number of its leased SDs. Thus, it might install several applications into the same SD for the service providers that are not willing to pay the cost of having their own SD.

Synchronization. The TSM accepts to continue sending the attacker queries without receiving any acknowledgment. As a matter of fact, this mode of asynchronous communication is often employed for optimization. Indeed, the transmission rate of smart cards is slow [RE10]. Therefore, the TSM usually pushes all the commands to the terminal. The terminal forwards them to the associated card, and then collects all the returned values to send them back to the TSM. Such method of communication helps improve not only the communication time, bu also the

undetectability of the attack, since the attacker application needs not to secretly include a special mode to manage all the sent commands during a session of TaGa. The attacker just intercepts and drops them.

Low-Entropy Data. This requirement is essential to succeed the Try-and-Guess attack. Low-entropy data are not rare in the context of smart cards. First, applications on smart cards often process enumerated variables with limited choices. This includes variables representing numerical values (e.g. amount of money), since integers are generally encoded by two bytes in smart cards (JavaCard v2 [Che00] only supports signed `short` as numerical type). In practice, 4-byte PIN codes ($\leq 10,000$ choices) are also considered as low-entropy data. Second, despite the length of plaintext, the format used for numerous card applications is quite predictable. Data, like those of GP commands [Glo15a] and the EMV standard [EMV], are often structured with ASN.1 BER-TLV [ISO02]. Such a format contains at least two public bytes: the tag value and the data length which the adversary already knows. In addition, the padding in SCP02 is constant and public. We illustrate by an example. The attacker wants to know how much money the user has provisioned her payment application. If the payment application is GP compliant, the provisioning command will be the GP command `Store Data`. Thus, the plaintext to be recovered is of the form:

$$\texttt{Tag}(1 \text{ byte}) \,||\, \texttt{Length}(1 \text{ byte}) \,||\, \texttt{Value}(2 \text{ bytes}) \,||\, \texttt{Padding}(4 \text{ bytes})$$

Within these eight bytes, the only bytes to be guessed are those of the `Value`. Therefore, there are no more than $2^{15} = 32,768$ choices, due to the fact that money should always remain positive. In practice, much fewer queries are required, since specific amounts of money are often suggested for account recharge.

5.5 Secure Channel Protocol '3'

5.5.1 Description

SCP03, published as an amendment to card specification 2.2 [Glo14], defines a new set of cryptographic methods based on AES. Similar to SCP02, it requires secret keys and padding, since it relies on block ciphers. SCP03 uses the "Encrypt-then-MAC" (EtM) method in which the ciphertext is produced by encrypting the message and then appending the result of applying the MAC scheme to the encrypted message. Figure 5.2 depicts how SCP03 computes ciphertexts. Refer to Construction 5.1 for more details about SCP03.

Construction 5.1. [*SCP03 Algorithms for Encryption and Decryption*]
Let E_{k} be an l-block-cipher and let $\text{CBC1}[E] = (\mathcal{K}\text{-CBC1}, n\mathcal{E}\text{-CBC1}, n\mathcal{D}\text{-CBC1})$ be its associated one-key nonce-based CBC encryption scheme (refer to Definition 3.4). Let $\mathcal{MA} = (\mathcal{K}_m, \mathcal{T}, \mathcal{V})$ be a message authentication scheme and let `Len` be a function returning the length of its input. For the sake of clarity, we do not include details about padding in the described algorithms. Given two independent keys $\mathsf{K1}$ and $\mathsf{K2}$ as well as a plaintext message $m \in \{0,1\}^{ln}$, where $n \in \mathbb{N}$, the scheme $\mathcal{SE}\text{-SCP03} = (\mathcal{K}\text{-SCP03}, \mathcal{E}\text{-SCP03}, \mathcal{D}\text{-SCP03})$ is defined as follows:

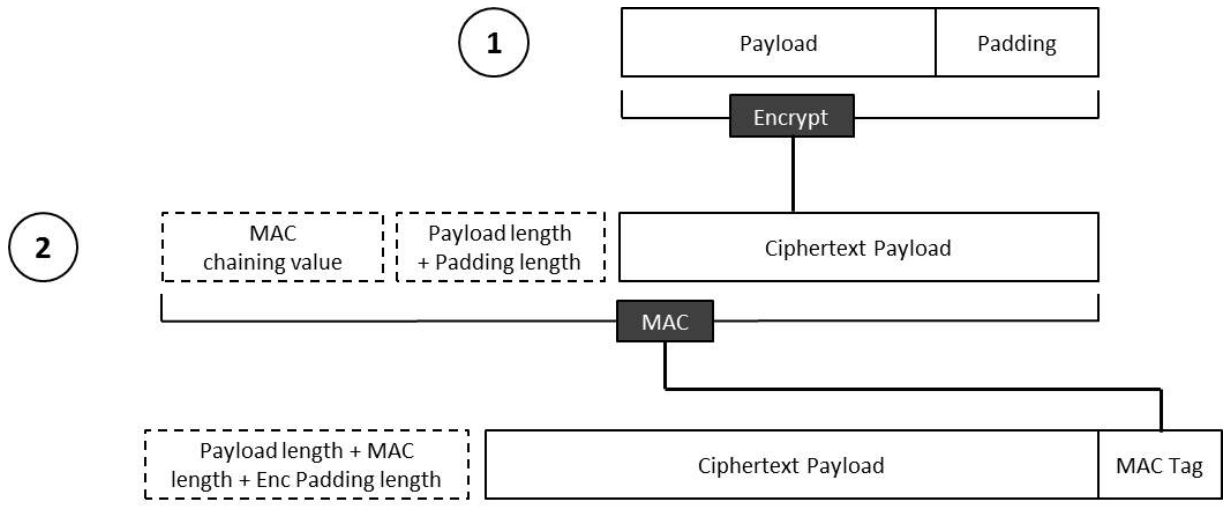


Figure 5.2: Ciphertext Generation by SCP03.

SCP03 Encryption $\mathcal{E}_{\mathsf{k}}$-SCP03(m)

1: $c \longleftarrow n\mathcal{E}_{\mathsf{k1}}$-CBC1$(\texttt{counter}, m)$
2: $\texttt{counter} \longleftarrow \texttt{counter} + 1$
3: $c' \longleftarrow \texttt{Len}(c) \,||\, c$
4: $\tau_1 || \tau_2 \longleftarrow \mathcal{T}_{\mathsf{k2}}(\texttt{chaining} \,||\, c')$
5: $\texttt{chaining} \longleftarrow \tau_1 || \tau_2$
6: **return** $c' \,||\, \tau_1$

SCP03 Decryption $\mathcal{D}_{\mathsf{k}}$-SCP03(c)

1: Parse c as $\texttt{Len}(c') \,||\, c' \,||\, \tau$
2: **if** cannot parse **then return** $\perp$
3: $c'' \longleftarrow \texttt{chaining} \,||\, \texttt{Len}(c') \,||\, c'$
4: $\tau_1 || \tau_2 \longleftarrow \mathcal{T}_{\mathsf{k2}}(c'')$
5: **if** $\tau_1 \neq \tau$ **then**
6: **return** $\perp$ **and** halt
7: **end if**
8: $\texttt{chaining} \longleftarrow \tau_1 || \tau_2$
9: $\texttt{ctxt} \longleftarrow n\mathcal{D}_{\mathsf{k1}}$-CBC1$(\texttt{counter}, c')$
10: $\texttt{counter} \longleftarrow \texttt{counter} + 1$
11: **return** $\texttt{ctxt}$

where `counter` and `chaining` are two static variables (i.e. maintain value between calls) that were properly initialized with predefined values. The `counter` variable, which is an integer, is initialized with $\mathbf{1}$, and the `chaining` variable, which is a string, is initialized with $\mathbf{0}^*$.

We highlight four points in the construction above. First, SCP03 ensures that all the message inputs to $\mathcal{T}_{\mathsf{k2}}$ and $\mathcal{V}_{\mathsf{k2}}$ are encoded. The encoding consists of appending the length of the input (i.e. c). Such encoding makes the set of inputs 'prefix-free', which means that no input can be the prefix of another one. This is an important requirement, since many MAC schemes, like CBC-MAC [BKR00], are secure only for prefix-free set of inputs. Second, we notice that SCP03 ends the opened secure session when a decryption fails. This approach of "halting state" makes SCP03 vulnerable to denial-of-service attacks. However, it is effective against chosen-ciphertext attacks, since all the ensuing ciphertexts will not be decrypted, and therefore a new session with new keys has to be re-negotiated. This makes such attacks more detectable and less likely to succeed. Third, we do not include the padding method of SCP03 (recommended in ISO/IEC 10116:2006 [ISO06]), since authors in [BU02] prove that it withstands padding oracle attacks (see their analysis of the OZ-PAD padding method). More importantly, Paterson et al. prove that padding has no negative impact on security when it is used in encryption schemes following the Encrypt-then-MAC (EtM) construction [PW12]. Fourth, the MAC construction is quite peculiar: only half of the MAC (i.e. 8 bytes) is included with the ciphertext, and the remainder is reconstructed during MAC verification. The other half is somehow used as a 'state' between

the sender and the receiver. To the best of our knowledge, GlobalPlatform has never provided the rationale behind this unusual construction that complicates the analysis of SCP03. However, we can plausibly assume that this choice was made to reduce the communication overhead incurred by SCP03. Indeed, the transmission rate with the card is low and it greatly increases with respect to the number of the communicated packets (as a matter of fact, the packet length is limited to 255 bytes) [RE10]. Therefore, despite being so small in other contexts, the overhead of transferring some extra 8 bytes might not be negligible in the case of smart cards.

5.5.2 Security Models

At first glance, SCP03 seems to fall into the EtM paradigm. Naturally, this raises no question regarding its security, since its generic security is proved in [BN08]. Here, we prove that SCP03 offers more than the standard security notions, namely IND-CPA and INT-CTXT (and hence IND-CCA).

Security Notions

A new concrete security treatment is required in order to capture the full power of SCP03. Here, we outline the security concepts that we will use to study SCP03.

Definition 5.1. [*Stateful Pseudorandom Function (sPRF)*]
Let $F = \{F_{\mathsf{k}} : \mathsf{K} \in \mathcal{K}(k)\}$ where F_{k} is a deterministic stateful function mapping l-bit strings to l'-bit strings for each $\mathsf{K} \in \mathcal{K}(k)$. let $R_{\mathcal{S}}$ be a stateful random-bit oracle. This means that the output of $R_{\mathcal{S}}$ depends on its state. Indeed, given a message $m \in \{0,1\}^l$, $R_{\mathcal{S}}(m)$ returns two different l'-bit random strings for two subsequent calls. The goal is that no adversary can distinguish whether she is interacting with a random instance of F or with the oracle $R_{\mathcal{S}}$. Stated more formally, F is said sPRF secure if for all polynomial-time distinguishers D, there is a negligible function negl such that:

$$\mathbf{Adv}^{\mathrm{sPRF}}_{\mathsf{D},F}(k) = \left|\mathsf{Pr}[\mathsf{D}^{F} = 1] - \mathsf{Pr}[\mathsf{D}^{R_{\mathcal{S}}} = 1]\right| \leq \mathsf{negl}(k)$$

Definition 5.2. [*Indistinguishability from tweakable random bits under CPA ($\widetilde{\text{IND}}$-CPA)*]
Here, we present a variant of the distinguishing concept defined for tweakable functions and presented in [BRW04]. We define $\widetilde{\text{IND}}$-CPA as follows. Let $F = \{F_{\mathsf{k}} : \mathsf{K} \in \mathcal{K}(k)\}$ where $F_{\mathsf{k}}(.,.)$ is a tweakable function mapping pairs of (t,l)-bit strings to l'-bit strings for each $\mathsf{K} \in \mathcal{K}(k)$. Let $\widetilde{\mathcal{R}}$ be a tweakable random-bit oracle from $\{0,1\}^t \times \{0,1\}^l$ to $\{0,1\}^{l'}$. We say that F is $\widetilde{\text{IND}}$-CPA secure if all polynomial-time distinguishers D, there is a negligible function negl such that:

$$\mathbf{Adv}^{\widetilde{\text{ind}}\text{-cpa}}_{\mathsf{D},F}(k) = \left|\mathsf{Pr}[\mathsf{D}^{F} = 1] - \mathsf{Pr}[\mathsf{D}^{\widetilde{\mathcal{R}}} = 1]\right| \leq \mathsf{negl}(k)$$

Definition 5.3. [*Indistinguishability under stateful CCA (IND-SFCCA)*]
This notion is defined by Bellare et al. in [BKN02]. Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. Let $\mathcal{A}$ be a polynomial-time adversary who has access to two oracles: a *left-or-right* encryption oracle $\mathcal{E}_{\mathsf{k}}(\mathcal{LR}(.,.,b))$ and a decryption oracle $\mathcal{D}_{\mathsf{k}}(.)$. The adversary $\mathcal{A}$ submits queries of the form (m_0, m_1) to the encryption oracle $\mathcal{E}_{\mathsf{k}}(\mathcal{LR}(.,.,b))$, where m_0 and m_1 must be of the same length (i.e. $|m_0| = |m_1|$). $\mathcal{A}$ is prohibited from querying the oracle $\mathcal{D}_{\mathsf{k}}(.)$ on a ciphertext that has already been output by the encryption oracle. Let i, j `sync` be three static variables that maintain their values between calls. For $k \in \mathbb{N}$, the following experiment is defined:

Experiment $\mathbf{Exp}^{\text{ind-sfcca-b}}_{\mathcal{SE},\mathcal{A}}(k)$

```
i ⟵ 1; j ⟵ 1
sync ⟵ true
repeat
    if A queries (m_0, m_1) then
        i ⟵ i + 1
        c_i ⟵ E_k(LR(m_0, m_1, b))
        output c_i to A
    end if
    if A queries c then
        j ⟵ j + 1
        m ⟵ D_k(c)
        if (j > i) or (c ≠ c_i) then
            sync ⟵ false
        end if
        if sync = false then
            output m to A
        end if
    end if
until A outputs b'
return b'
```

From the above experiment, we highlight the following point: $\mathcal{D}_{\mathsf{k}}$ returns the result of the decryption only when $\mathcal{A}$ makes an *out-of-sync* query. A query is out-of-sync if it satisfies one of these conditions: (1) there are more queries to the decryption oracle than to the encryption one; (2) the ciphertext inside the decryption query is different from the last one computed by $\mathcal{E}_{\mathsf{k}}(\mathcal{LR}(.,.,b))$. As long as $\mathcal{A}$ does not make out-of-sync queries, $\mathcal{D}_{\mathsf{k}}$ returns nothing and just updates its internal state.

We say that $\mathcal{SE}$ is IND-SFCCA secure if for all polynomial-time adversaries $\mathcal{A}$, there is a negligible function negl such that:

$$\mathbf{Adv}^{\text{ind-sfcca}}_{\mathcal{A},\mathcal{SE}}(k) = \left|\Pr[\mathbf{Exp}^{\text{ind-sfcca-1}}_{\mathcal{SE},\mathcal{A}}(k) = 1] - \Pr[\mathbf{Exp}^{\text{ind-sfcca-0}}_{\mathcal{SE},\mathcal{A}}(k) = 1]\right| \leq \mathsf{negl}(k)$$

Definition 5.4. [*Integrity of stateful ciphertext (INT-SFCTXT)*]
This notion is defined by Bellare et al. in [BKN02]. Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. Let $\mathcal{A}$ be a polynomial-time adversary who has access to two oracles: an encryption oracle $\mathcal{E}_{\mathsf{k}}(.)$ and a decryption oracle $\mathcal{D}_{\mathsf{k}}(.)$. Let S be the list of all ciphertexts generated by the adversary queries to $\mathcal{E}_{\mathsf{k}}(.)$. Let i, j `sync` be three static variables that maintain their values between calls. For $k \in \mathbb{N}$, the following experiment is defined:

Experiment $\mathbf{Exp}^{\text{ind-sfctxt}}_{\mathcal{SE},\mathcal{A}}(k)$

```
i ⟵ 1; j ⟵ 1
sync ⟵ true
repeat
    if A queries m then
        i ⟵ i + 1
        c_i ⟵ E_k(m)
        output c_i to A
    end if
    if A queries c then
        j ⟵ j + 1
        m ⟵ D_k(c)
        if (j > i) or (c ≠ c_i) then
            sync ⟵ false
        end if
        if sync = false then
            output m to A
        end if
    end if
until A outputs c'
if D_k(c') ≠ ⊥ and c' ∉ S then
    return 1
else
    return 0
end if
```

We say that $\mathcal{SE}$ is INT-SFCTXT secure if for all polynomial-time adversaries $\mathcal{A}$, there is a negligible function negl such that:

$$\mathbf{Adv}^{\text{int-sfctxt}}_{\mathcal{A},\mathcal{SE}}(k) = \mathsf{Pr}[\mathbf{Exp}^{\text{int-sfctxt}}_{\mathcal{SE},\mathcal{A}}(k) = 1] \leq \mathsf{negl}(k)$$

Definition 5.5. [*Algorithm-Substitution Attacks (ASA)*]
Motivated by the potential threat of subverting implementations of cryptographic algorithms, Bellare, Paterson and Rogaway in [BPR14] have recently defined ASA security by identifying two adversarial goals – *conducting surveillance* and *avoid detection*. In the ASA experiment, given user's key K and a subversion key $\widetilde{\mathsf{K}}$, the adversary $\mathcal{B}$ (also called big brother) wants to subvert the encryption algorithm $\mathcal{E}_{\mathsf{k}}$ by another one $\widetilde{\mathcal{E}}_{\widetilde{\mathsf{k}}}$. $\mathcal{B}$ requires that the subversion be both successful and undetectable. Here, we focus solely on the surveillance goal (SURV). SURV means that from observing ciphertexts, $\mathcal{B}$ can compromise confidentiality. Stated formally, SURV is defined as a classical distinguishing experiment when given oracle access to one of these two algorithms (i.e. $\mathcal{E}_{\mathsf{k}}$ and $\widetilde{\mathcal{E}}_{\widetilde{\mathsf{k}}}$). Indeed, $\mathcal{B}$, who has access to K but not to $\widetilde{\mathsf{K}}$, is required to distinguish $\mathcal{E}_{\mathsf{k}}$ from $\widetilde{\mathcal{E}}_{\widetilde{\mathsf{k}}}$. We say that an encryption scheme is ASA secure if no polynomial-time adversary $\mathcal{B}$ can succeed the SURV distinguishing game. As usual, we say that $\mathcal{SE}$ is SURV secure if for all polynomial-time adversaries $\mathcal{B}$, there is a negligible function negl such that:

$$\mathbf{Adv}^{\text{surv}}_{\mathcal{B},\mathcal{SE}}(k) = \left|\mathsf{Pr}[\mathcal{B}^{\mathcal{E}_{\mathsf{k}}} = 1] - \mathsf{Pr}[\mathcal{B}^{\widetilde{\mathcal{E}}_{\widetilde{\mathsf{k}}}} = 1]\right| \leq \mathsf{negl}(k)$$

Definition 5.6. [*Unique Ciphertexts (UQ-CTXT)*]
Following their work to defeat ASA, Bellare et al. define the notion of 'Unique Ciphertexts' as follows. Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. Given a secret key k, a message m, and a state τ, let $\mathcal{C}_{\mathcal{SE}}(\mathsf{k}, m, \tau)$ be the set of all ciphertexts such that $\mathcal{D}^{\tau}_{\mathsf{k}}(c)$ (also denoted

$\mathcal{D}_{\mathsf{k}}(c_\tau)$) returns m. We say that $\mathcal{SE}$ has unique ciphertexts (i.e. UQ-CTXT secure) if the set $\mathcal{C}_{\mathcal{SE}}(\mathsf{k}, m, \tau)$ has size at most one for all k, m, τ. Stated differently, for any given key, message and state, there exists at most one ciphertext that decrypts to the message in question.

Result 5.1. [*Unique Ciphertexts* $\implies$ *ASA resilience [BPR14]*]
In other words, let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a unique ciphertext encryption scheme, and let $\mathcal{B}$ be a SURV adversary. Then, $\mathcal{B}$ cannot succeed the SURV experiment; which means that $\mathcal{SE}$ is resilient to ASA.

Analyzing SCP03 via a New Construction

The construction of SCP03 described in 5.5.1 brings out three points that should be underlined. First, both the encryption and decryption algorithms involve the use of two variables that maintain their values and get updated after each call. These two variables must be '*in-sync*' between the sender and the receiver, otherwise $\mathcal{D}_{\mathsf{k}}$-SCP03(.) returns $\perp$. Second, the encryption of messages could be seen as a stateful nonce-based CBC encryption scheme. Third, the `chaining` variable serves much the same purpose that a tweak does. Taking into consideration these three notes, we can turn the EtM construction of SCP03 into another composite. We start by introducing the two underlying blocks that will compose our new equivalent construction of SCP03.

Definition 5.7. [*Stateful Nonce-based Symmetric Scheme (Sf-n$\mathcal{SE}$)*]
Let $n\mathcal{SE} = (\mathcal{K}, n\mathcal{E}, n\mathcal{D})$ be a nonce-based encryption scheme. Let `counter` be a static variable that is always initialized by the same predefined value and which maintains its value between calls. Given a message m, we define the associated stateful scheme Sf-$n\mathcal{SE} = (\mathcal{K}\text{-Sf}, n\mathcal{E}\text{-Sf}, n\mathcal{D}\text{-Sf})$ as follows:

Sf-n$\mathcal{SE}$ Encryption $n\mathcal{E}$-$\mathrm{Sf}_{\mathsf{k}}(m)$

1: $c \longleftarrow n\mathcal{E}_{\mathsf{k}}(m)$
2: `counter` $\longleftarrow$ `counter` $+ 1$
3: **return** c

Sf-n$\mathcal{SE}$ Decryption $n\mathcal{D}$-$\mathrm{Sf}_{\mathsf{k}}(c)$

1: $c \longleftarrow n\mathcal{D}_{\mathsf{k}}(c)$
2: `counter` $\longleftarrow$ `counter` $+ 1$
3: **return** m

Definition 5.8. [*Tweak Chaining MAC ($\mathcal{TC}$-$\widetilde{\mathcal{MA}}$)*]
Let $\widetilde{F}_{\mathsf{k}} : \{0,1\}^t \times \{0,1\}^* \longleftarrow \{0,1\}^t$ be a tweakable MAC function for all key $\mathsf{K} \in$ `Key`. Let `chaining` be a static variable that is always initialized by the same predefined value and which maintains its value between calls. Then, we define the associated chaining scheme $\mathcal{TC}$-$\widetilde{\mathcal{MA}} = (\widetilde{\mathcal{K}}, \widetilde{\mathcal{T}}, \widetilde{\mathcal{V}})$ as follows:

$\mathcal{TC}$-$\widetilde{\mathcal{MA}}$ **Tagging** $\widetilde{\mathcal{T}}_{\mathsf{k}}(m)$

1: $\tau_1 || \tau_2 \longleftarrow \widetilde{F}_{\mathsf{k}}(\texttt{chaining}, m)$
2: `chaining` $\longleftarrow \tau_1 || \tau_2$
3: **return** τ_1

$\mathcal{TC}$-$\widetilde{\mathcal{MA}}$ **Verification** $\widetilde{\mathcal{V}}_{\mathsf{k}}(m, \tau)$

1: $\tau_1 || \tau_2 \longleftarrow \widetilde{F}_{\mathsf{k}}(\texttt{chaining}, m)$
2: $b \longleftarrow [\tau_1 = \tau]$
3: `chaining` $\longleftarrow \tau_1 \,||\, \tau_2$
4: **return** b

Construction 5.2. [*Stateful Nonce-based Encrypt-then-Tweak (Sf-nEtTw)*]
Let Sf-$n\mathcal{SE} = (\mathcal{K}\text{-Sf}, n\mathcal{E}\text{-Sf}, n\mathcal{D}\text{-Sf})$ be a stateful nonce-based symmetric scheme. Let $(\mathsf{Enc}, \mathsf{Dec})$ be a prefix-free encoding scheme. Let $\mathcal{TC}$-$\widetilde{\mathcal{MA}} = (\widetilde{\mathcal{K}}, \widetilde{\mathcal{T}}, \widetilde{\mathcal{V}})$ be a tweak chaining MAC. Given a

message m and a key $\mathsf{K} = \mathsf{K1} \,||\, \mathsf{K2}$, where $\mathsf{K1}$ and $\mathsf{K2}$ are two independent keys, we define the composite stateful nonce-based Encrypt-then-Tweak AE scheme **Sf-nEtTw** $= (\widetilde{\mathcal{K}}\text{-Sf}, \widetilde{\mathcal{E}}\text{-Sf}, \widetilde{\mathcal{D}}\text{-Sf})$ as follows:

Sf-nEtTw Encryption $\widetilde{\mathcal{E}}_{\mathsf{k}}\text{-Sf}(m)$

1: $c \longleftarrow n\mathcal{E}_{\mathsf{k1}}\text{-Sf}(m)$
2: $c' \longleftarrow \mathsf{Enc}(c)$
3: $\tau \longleftarrow \widetilde{\mathcal{T}}_{\mathsf{k2}}(c')$
4: **return** $c' \,||\, \tau$

Sf-nEtTw Decryption $\widetilde{\mathcal{D}}_{\mathsf{k}}\text{-Sf}(c)$

1: Parse c as $c' \,||\, \tau$
2: $c'' \longleftarrow \mathsf{Dec}(c')$ or **return** $\perp$
3: **if** $\widetilde{\mathcal{V}}_{\mathsf{k2}}(c', \tau) \neq 1$ **then**
4: **return** $\perp$ **and** halt
5: **end if**
6: **return** $n\mathcal{D}_{\mathsf{k1}}\text{-Sf}(c'')$

Let us see if the Construction 5.1 actually implies the definition of Construction 5.2. We start by examining whether the SCP03 operation $\widetilde{\mathcal{T}}_{\mathsf{k}}(\texttt{chaining} \,||\, c)$ is indeed a secure tweakable MAC function. We notice that the MAC computation in SCP03 is based on CMAC as specified in [Dwo01b]. As mentioned by the author, CMAC is equivalent to OMAC [IK03]. We rely on the result of [BRW04] in which authors prove that $\widetilde{\text{OMAC}}(t \,||\, m)$ is an IND-CPA tweakable extension of OMAC. Hence, we have $\widetilde{\mathcal{T}}_{\mathsf{k}}(\texttt{chaining} \,||\, c) = \widetilde{F}_{\mathsf{k}}(\texttt{chaining}, c)$, where $\widetilde{F}_{\mathsf{k}}$ is a MAC tweakable function.

Now, we investigate the security of the SCP03 encryption scheme that could be seen as a stateful variant of CBC1. Defined in section 3.4, we recall that CBC1 is a nonce-based scheme that encrypts the nonce to use it as IV. It was recommended by NIST in 2001 [Dwo01a] and was broken three years later in [Rog04b]. Unlike the insecure CBC1, the stateful CBC1 is IND-CPA secure. The intuition behind this is that attacks against CBC1 generally involve a craftily chosen nonce, and therefore they are not applicable against the stateful CBC1 where nonces are taken as a counter. The full proof is given in Theorem 17 in [BDJR97].

5.6 SCP03 Security Results

We now provide our security results regarding SCP03. We provably show that SCP03 protects the integrity and the confidentiality of messages against chosen-plaintext and chosen-ciphertext attacks. Stated more formally, SCP03 is both IND-CPA and INT-CTXT secure, and therefore IND-CCA secure. In addition, we prove that it also resists replay, out-of-delivery and algorithm-substitution attacks (ASAs). Indeed, authors of [BKN02, BPR14] have proved that cryptographic schemes satisfying IND-SFCCA, INT-SFCTXT and Unique Ciphertexts meet all the security notions mentioned above.

5.6.1 Sf-nEtTw Security Analysis

We start by analyzing the composite encryption scheme Sf-nEtTw. The following proposition concerns the security properties of $\mathcal{TC}\text{-}\widetilde{\mathcal{MA}}$.

Proposition 5.1. [*Upper Bound of* $\mathbf{Adv}^{\textit{suf-cma}}_{\mathcal{A}, \mathcal{TC}\text{-}\widetilde{\mathcal{MA}}}(k)$]
Let $\widetilde{F}_{\mathsf{k}} : \{0,1\}^n \times \{0,1\}^* \longrightarrow \{0,1\}^n$ be a tweakable function and let $\mathcal{TC}\text{-}\widetilde{\mathcal{MA}}$ be its associated chaining scheme. Consider any polynomial-time SUF-CMA adversary $\mathcal{A}$ attacking $\mathcal{TC}\text{-}\widetilde{\mathcal{MA}}$ and querying q messages. Then, we can construct a distinguisher D against $\widetilde{F}$ such that:

$$\mathbf{Adv}^{\text{suf-cma}}_{\mathcal{A}, \mathcal{TC}\text{-}\widetilde{\mathcal{MA}}}(k) \leq \mathbf{Adv}^{\widetilde{\text{ind-cpa}}}_{\mathsf{D}, \widetilde{F}}(k) + \frac{q^2}{2^n} + \frac{1}{2^{n/2}} + \mathsf{Pr}[\mathsf{Col}_q]$$

where $\mathsf{Pr}[\mathsf{Col}_q]$ is the collision probability of the tweakable function $\widetilde{F}_{\mathsf{k}}$ after q messages.

Proof. Due to its technicality, we defer the proof to section 5.7.1. □

We now show how schemes following the construction of Sf-nEtTw protect their stateful integrity of ciphertext (i.e. INT-SFCTXT).

Theorem 5.1. [*Upper Bound of* $\mathbf{Adv}^{int\text{-}sfctxt}_{\mathcal{A},\, Sf\text{-}nEtTw}(k)$]
Let Sf-nEtTw be a scheme of stateful nonce-based encryption Sf-$n\mathcal{SE}$ = ($\mathcal{K}$-Sf, $n\mathcal{E}$-Sf, $n\mathcal{D}$-Sf) associated to a tweak chaining MAC $\mathcal{TC}$-$\widetilde{\mathcal{MA}} = (\widetilde{\mathcal{K}}, \widetilde{\mathcal{T}}, \widetilde{\mathcal{V}})$ and a prefix-free encoding scheme (Enc, Dec) as described in Construction 5.2. Let $\widetilde{F}_{\mathsf{k}} : \{0,1\}^n \times \{0,1\}^* \longrightarrow \{0,1\}^n$ be the tweakable MAC function related to $\mathcal{TC}$-$\widetilde{\mathcal{MA}}$. Consider any INT-SFCTXT adversary $\mathcal{A}$ against Sf-nEtTw who asks to encrypt q messages, we can construct an SUF-CMA adversary $\mathcal{B}$ against $\mathcal{TC}$-$\widetilde{\mathcal{MA}}$ such that:

$$\mathbf{Adv}^{\text{int-sfctxt}}_{\mathcal{A},\,\text{Sf-nEtTw}}(k) \leq \mathbf{Adv}^{\text{suf-cma}}_{\mathcal{B},\,\mathcal{TC}\text{-}\widetilde{\mathcal{MA}}}(k) + \mathsf{Pr}[\mathsf{q\text{-}Col}]$$

where, given a message m and a list S containing q outputs of $\widetilde{F}_{\mathsf{k}}$, $\mathsf{Pr}[\mathsf{q\text{-}Col}]$ is the probability that $\widetilde{F}_{\mathsf{k}}(m) \in \mathsf{S}$.

Proof. Due to its technicality, we defer the proof to section 5.7.2. □

5.6.2 SCP03 Security Analysis

OMAC Collision Probabilities

Now, we give our concrete security results for the particular case of SCP03. This requires to compute the different collision probabilities when $\widetilde{\text{OMAC}}_{\mathsf{k}}(T, m) = \text{OMAC}_{\mathsf{k}}(T \,||\, m)$ is used as the tweakable function $\widetilde{F}_{\mathsf{k}}(.,.)$ for all tweak T and message m. Hence, we start by stating two Results proved in [IK03] about collisions in OMAC.

Result 5.2. [$\mathsf{Pr}[\mathsf{Col}_2]$]
Let E_{k} be a block cipher of size l and let OMAC$[E_{\mathsf{k}}]$ be its associated OMAC scheme. For the sake of simplicity, we only consider messages m whose length is a multiple of l (i.e. $|m|/l$ is an integer). Given a message m, we denote by μ the number of its blocks, namely $\mu = |m|/l$. Consider two messages m and m', then the following relation characterizes the probability of the OMAC collision:

$$\mathsf{Pr}[\mathsf{Col}_2] = \mathsf{Pr}[\mathsf{Col}(m, m')] \leq \frac{(\mu + \mu')^2}{2^l}$$

Result 5.3. [$\mathsf{Pr}[\mathsf{Col}_q]$]
Let E_{k} be a block cipher of size l and let OMAC$[E_{\mathsf{k}}]$ be its associated OMAC scheme. For the sake of simplicity, we only consider messages m whose length is a multiple of l (i.e. $|m|/l$ is an integer). Given a message m, we denote by μ the number of its blocks, namely $\mu = |m|/l$. Given a list $\mathcal{Q}$ of q messages, the following relation characterizes the probability of the OMAC collision on $\mathcal{Q}$:

$$\mathsf{Pr}[\mathsf{Col}_q] = \mathsf{Pr}[\mathsf{Col}(\mathcal{Q})] \leq \frac{\left(\sum_{i=1}^{q} \mu_i\right)^2}{2^l}$$

SCP03 is both INT-SFCTXT and IND-SFCCA

We now show that SCP03 protects its stateful confidentiality and integrity against powerful adversaries who can perform chosen-ciphertext attacks (CCA).

Theorem 5.2. [*SCP03 is INT-SFCTXT Secure*]
Let E_{k} be a block cipher of size n and let $\mathrm{OMAC}[E_{\mathsf{k}}](.)$ be its associated OMAC scheme. Let $\widetilde{\mathrm{OMAC}}_{\mathsf{k}}(.,.)$ be a tweakable function defined as $\widetilde{\mathrm{OMAC}}_{\mathsf{k}}(T,m) = \mathrm{OMAC}[E_{\mathsf{k}}](T \,||\, m)$ for all tweak T and message m. Given a prefix-free encoding scheme $(\mathsf{Enc}, \mathsf{Dec})$, a stateful nonce-based encryption $\text{Sf-}n\mathcal{SE} = (\mathcal{K}\text{-Sf}, n\mathcal{E}\text{-Sf}, n\mathcal{D}\text{-Sf})$ and a tweak chaining MAC $\mathcal{TC}\text{-}\widetilde{\mathcal{MA}} = (\widetilde{\mathcal{K}}, \widetilde{\mathcal{T}}, \widetilde{\mathcal{V}})$ whose tweakable MAC function is $\widetilde{\mathrm{OMAC}}_{\mathsf{k}}$, we define SCP03 to be the composite scheme formed by following the Construction 5.2. Consider any INT-SFCTXT polynomial-time adversary $\mathcal{A}$ attacking SCP03 and asking to encrypt q messages, we can construct a distinguisher D against $\widetilde{\mathrm{OMAC}}$ and a negligible function negl such that:

$$\mathbf{Adv}^{\text{int-sfctxt}}_{\mathcal{A},\,\text{scp03}}(k) \leq \mathbf{Adv}^{\widetilde{\text{ind-cpa}}}_{\mathsf{D},\,\widetilde{\mathrm{OMAC}}}(k) + \mathsf{negl}(k)$$

Proof. Since SCP03 is a composite scheme formed by following the Sf-nEtTw construction, it satisfies the relations given in section 5.6.1. By using Proposition 5.1 and Theorem 5.1, we can obtain that

$$\mathbf{Adv}^{\text{int-sfctxt}}_{\mathcal{A},\,\text{scp03}}(k) \leq \mathbf{Adv}^{\widetilde{\text{ind-cpa}}}_{\mathsf{D},\,\widetilde{\mathrm{OMAC}}}(k) + \frac{q^2}{2^n} + \frac{1}{2^{n/2}} + \mathsf{Pr}[\mathsf{Col}_q] + \mathsf{Pr}[\mathsf{q\text{-}Col}]$$

where $\mathsf{Pr}[\mathsf{Col}_q]$ is the collision probability of the tweakable function $\widetilde{\mathrm{OMAC}}_{\mathsf{k}}$ after q messages and $\mathsf{Pr}[\mathsf{q\text{-}Col}]$, given a message m and a list S containing q outputs of $\widetilde{\mathrm{OMAC}}_{\mathsf{k}}$, is the probability that $\widetilde{\mathrm{OMAC}}_{\mathsf{k}}(m) \in \mathsf{S}$.

Now, we use the following lemma to conclude our proof. □

Lemma 5.1. Given $n \in \mathbb{N}$, there is a negligible function negl such that:

$$\frac{q^2}{2^n} + \frac{1}{2^{n/2}} + \mathsf{Pr}[\mathsf{Col}_q] + \mathsf{Pr}[\mathsf{q\text{-}Col}] \leq \mathsf{negl}$$

where $\mathsf{Pr}[\mathsf{Col}_q]$ and $\mathsf{Pr}[\mathsf{q\text{-}Col}]$ are as defined above.

Proof. We need to compute both $\mathsf{Pr}[\mathsf{Col}_q]$ and $\mathsf{Pr}[\mathsf{q\text{-}Col}]$. The case of $\mathsf{Pr}[\mathsf{Col}_q]$ is easy and it can be immediately obtained from Result 5.3. Concerning the case $\mathsf{Pr}[\mathsf{q\text{-}Col}]$, it can be calculated from Result 5.2. Indeed, given a message m and a list S, $\mathsf{Pr}[\mathsf{q\text{-}Col}]$ can be expressed as the sum of the collision probabilities $\mathsf{Pr}[\mathsf{Col}_2^i]$ between m and a message m_i for all $m_i \in \mathsf{S}$. Here, $m_{\max}$ denotes any message of maximum length and $\mu_{\max}$ denotes its number of blocks. Therefore, we have

$$\mathsf{Pr}[\mathsf{q\text{-}Col}] \leq \sum_{i=1}^{q} \mathsf{Pr}[\mathsf{Col}(m_{\max}, m_i)] \leq \sum_{i=1}^{q} \frac{(\mu_{\max} + \mu_i)^2}{2^n}$$

From all the relations above, we have

$$\begin{aligned}
\frac{q^2}{2^n} + \frac{1}{2^{n/2}} + \mathsf{Pr}[\mathsf{Col}_q] + \mathsf{Pr}[\mathsf{q\text{-}Col}] &\leq \frac{1}{2^{n/2}} + \frac{q^2}{2^n} + \frac{\left(\sum_{i=1}^{q} \mu_i\right)^2}{2^n} + \frac{\sum_{i=1}^{q} (\mu_{\max} + \mu_i)^2}{2^n} \\
&\leq \frac{1}{2^{n/2}} + \frac{q^2}{2^n} + \frac{\left(\sum_{i=1}^{q} \mu_i\right)^2}{2^n} + \frac{3q\mu_{\max}^2 + \sum_{i=1}^{q} \mu_i^2}{2^n} \\
&\leq \frac{1}{2^{n/2}} + \frac{q\left(q + (2\mu_{\max})^2\right)}{2^n} + \frac{\left(\sum_{i=1}^{q} \mu_i\right)^2 + \sum_{i=1}^{q} \mu_i^2}{2^n}
\end{aligned}$$

which is asymptotically negligible. □

Theorem 5.3. [*SCP03 is IND-SFCCA Secure*]
Let E_{k} be a block cipher and let OMAC$[E_{\mathsf{k}}](.)$ be its associated OMAC scheme. Let $\widetilde{\text{OMAC}}_{\mathsf{k}}(.,.)$ be a tweakable function defined as $\widetilde{\text{OMAC}}_{\mathsf{k}}(T,m) = \text{OMAC}[E_{\mathsf{k}}](T \,||\, m)$ for all tweak T and message m. Given a prefix-free encoding scheme $(\mathsf{Enc}, \mathsf{Dec})$, a stateful nonce-based encryption Sf-$n\mathcal{SE} = (\mathcal{K}\text{-Sf}, n\mathcal{E}\text{-Sf}, n\mathcal{D}\text{-Sf})$ and a tweak chaining MAC $\mathcal{TC}\text{-}\widetilde{\mathcal{MA}} = (\widetilde{\mathcal{K}}, \widetilde{\mathcal{T}}, \widetilde{\mathcal{V}})$ whose tweakable MAC function is $\widetilde{\text{OMAC}}_{\mathsf{k}}$, we define SCP03 to be the composite scheme formed by following the Construction 5.2. Consider any IND-SFCCA polynomial-time adversary $\mathcal{A}$ against SCP03, we can construct an IND-CPA adversary $\mathcal{B}$ against Sf-$n\mathcal{SE}$ and an INT-SFCTXT adversary $\mathcal{F}$ against $\mathcal{TC}\text{-}\widetilde{\mathcal{MA}}$ such that:

$$\mathbf{Adv}^{\text{ind-sfcca}}_{\mathcal{A},\,\text{scp03}}(k) \leq \mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{B},\,\text{Sf-}n\mathcal{SE}}(k) + \mathbf{Adv}^{\text{int-sfctxt}}_{\mathcal{F},\,\mathcal{TC}\text{-}\widetilde{\mathcal{MA}}}(k)$$

Proof. This theorem follows directly from the implication proved by Bellare et al. [BKN02]: IND-CPA $\wedge$ INT-SFCTXT $\implies$ IND-SFCCA. This means that if an encryption scheme is both IND-CPA and INT-SFCTXT secure, then it is also IND-SFCCA secure. Regarding the INT-SFCTXT security of SCP03, we have just proved it in Theorem 5.2. Now, let us consider the IND-CPA security property of SCP03. Notice that SCP03 is a variant of Encrypt-then-MAC. Therefore, it inherits the IND-CPA property of its encryption scheme [BN08]. Stated otherwise, if the underlying encryption scheme Sf-$n\mathcal{SE}$ is IND-CPA secure, then SCP03 is also IND-CPA secure, which concludes our proof. □

SCP03 is ASA Resilient

Finally, we prove that SCP03 defends against ASA, hence also against mass surveillance.

Theorem 5.4. [*SCP03 has Unique Ciphertexts*]
Let $\widetilde{\text{OMAC}}_{\mathsf{k}}$ be a tweakable function as defined previously. Given a stateful nonce-based encryption Sf-$n\mathcal{SE} = (\mathcal{K}\text{-Sf}, n\mathcal{E}\text{-Sf}, n\mathcal{D}\text{-Sf})$ and a tweak chaining MAC $\mathcal{TC}\text{-}\widetilde{\mathcal{MA}} = (\widetilde{\mathcal{K}}, \widetilde{\mathcal{T}}, \widetilde{\mathcal{V}})$ whose tweakable MAC function is $\widetilde{\text{OMAC}}_{\mathsf{k}}$, we define SCP03 to be the composite scheme formed by following the Construction 5.2. Then, SCP03 is UQ-CTXT secure.

Proof. Let c_i denote the ciphertext produced by encrypting the message m on the state i. Considering the SCP03 design (see Construction 5.2), we have

$$c_i = \sigma_i \,||\, \tau_i = n\mathcal{E}_{\mathsf{k1}}\text{-Sf}(m) \,||\, \widetilde{\mathcal{T}}_{\mathsf{k2}}(\sigma_i)$$

where $\mathsf{K1}$ and $\mathsf{K2}$ are two independent keys. Now, we study the probability of finding a triplet $(\mathsf{k} = \mathsf{k1}||\mathsf{k2}, m, i)$ so that $|\mathcal{C}_{\text{scp03}}(\mathsf{k}, m, i)| > 1$. By definition, this is equal to the probability that of finding a ciphertext c'_i such that: (1) $c'_i \neq c_i$ and (2) $\mathcal{D}_{\mathsf{k}}\text{-SCP03}(c'_i) = m'$, where $m' = m$. We proceed by distinguishing two cases.

Case 1 ($\boldsymbol{\sigma'_i \neq \sigma i}$). Here, we prove that this case and the event of finding c' are contradictory, thereby proving that $\mathsf{Pr}[\mathsf{case\ 1}] = 0$. Indeed, recall that $n\mathcal{SE} = (\mathcal{K}, n\mathcal{E}, n\mathcal{D})$ encrypts messages using a deterministic algorithm. Therefore, as a matter of fact, for a fixed nonce n, $n\mathcal{E}_{\mathsf{k1}}(n, m_1) \neq n\mathcal{E}_{\mathsf{k1}}(n, m_2)$ implies that $m_1 \neq m_2$. Also, we notice that the definition of the set $\mathcal{C}_{\text{scp03}}$ involves that the associated encryption scheme Sf-$n\mathcal{SE} = (\mathcal{K}\text{-Sf}, n\mathcal{E}\text{-Sf}, n\mathcal{D}\text{-Sf})$ has called $n\mathcal{SE}$ algorithms with the same nonce for each state i. Then, the event $\sigma'_i \neq \sigma_i$ entails $\sigma'_i = n\mathcal{E}_{\mathsf{k1}}\text{-Sf}(m') \neq n\mathcal{E}_{\mathsf{k1}}\text{-Sf}(m) = \sigma_i$, which implies $m' \neq m$. This concludes our proof, since the definition of $\mathcal{C}_{\text{scp03}}$ includes that $m' = m$.

Case 2 ($\boldsymbol{\sigma'_i = \sigma_i}$). Since $c'_i \neq c_i$, this case implies that $\tau'_i \neq \tau_i$. Following the same argument of $\mathsf{case\ 1}$, we prove that this case and the event of finding c' are contradictory, thereby proving that

$\mathsf{Pr}[\mathsf{case\ 2}] = 0$. Indeed, recall that the tweakable MAC function $\widetilde{\mathrm{OMAC}}_{\mathsf{k2}}$ generates its tag using a deterministic algorithm. Therefore, as a matter of fact, for a fixed tweak t, $\widetilde{\mathrm{OMAC}}_{\mathsf{k2}}(t, \sigma_1) \neq \widetilde{\mathrm{OMAC}}_{\mathsf{k2}}(t, \sigma_2)$ implies that $\sigma_1 \neq \sigma_2$. Similarly, the definition of $\mathcal{C}_{\mathrm{scp03}}$ involves that the associated chaining MAC scheme $\mathcal{TC}\text{-}\widetilde{\mathcal{MA}} = (\widetilde{\mathcal{K}}, \widetilde{\mathcal{T}}, \widetilde{\mathcal{V}})$ has called $\widetilde{\mathrm{OMAC}}_{\mathsf{k2}}$ with same tweak for each state i. Then, the event $\tau_i' \neq \tau_i$ entails $\tau_i' = \widetilde{\mathcal{T}}_{\mathsf{k2}}(\sigma_i') \neq \widetilde{\mathcal{T}}_{\mathsf{k2}}(\sigma_i) = \tau_i$, which implies $\sigma_i' \neq \sigma_i$. This concludes our proof, since the definition of $\mathsf{case\ 2}$ includes that $\sigma_i' = \sigma_i$. □

5.6.3 Practical Considerations of SCP03 Security Analysis

Instead of just proving asymptotic results in our security analysis in section 5.6.2, we have attempted to explicitly capture the quantitative nature of SCP03 security. This brings a new dimension to our security analysis: we can *measure* how secure SCP03 is by computing how long SCP03 maintains its security in terms of the number and the size of encrypted messages. Indeed, the Theorem 5.2 characterizing the INT-SFCTXT security of SCP03 says that a polynomial-time adversary who sees q encrypted messages of μ_i blocks has chance of at most $\epsilon + 1/2^{\frac{n}{2}} + (1/2^n)(q^2 + 4q\mu_{\max}^2 + (\sum_{i=1}^q \mu_i)^2 + \sum_{i=1}^q \mu_i^2)$ of breaking the scheme, where n is the block size of the underlying block cipher (i.e. AES), $\mu_{\max}$ is the maximum number of blocks in a message (i.e. $\mu_{\max} = |m_{\max}|/n$), and ϵ captures the security of the tweakable variant of OMAC.

Recall that SCP03 relies on a counter-based encryption scheme and it ends the secure session whenever the counter wraps up and starts again from 0. This means that the size of the counter determines how long a secure session might last. However, there is no mention in the GlobalPlatform specification [Glo14] about the recommended size of the used counter. We might believe that the counter consists of 4 bytes due to the fact that integers are generally encoded with 4 bytes. We notice that specifications in the same document define a 3-byte counter that is incremented whenever a secure session is opened. That counter is associated to a master key and lives as long as its related key does. Therefore, we think that the SCP03 designers make it as big as possible. As a result, our assumption about the size of the counter used inside the encryption scheme (i.e. 4 bytes) seems reasonable. Thus, an adversary might make no more than 2^{32} queries during a secure session, which is more than enough for most card purposes. Henceforth, $q_{\max}$ denotes the maximum number of queries during a session.

By looking into the implementation details of SCP03, we see that the cipher block in use is AES, hence $n = 128$. In addition, we see that $\mu_{\max} = 16$, since SCP03 is defined to be used in the context of smart cards, and therefore a message in the SCP03 protocol represents an APDU. Recall that APDUs cannot be longer than 256 bytes (see [ISO13]).

Now, we consider an adversary making $q_{\max}$ queries, each of which consists of $\mu_{\max}$ blocks. Let us study the impact of such an adversary on the concrete security of SCP03. In order to simplify our computation, we express all the terms with respect to n: $q_{\max} = 2^{n/4}$ and $\mu_{\max} = 2^{n/8}$. Consequently, we have

$$
\begin{aligned}
&\frac{1}{2^{n/2}} + \frac{q\left(q + (2\mu_{\max})^2\right)}{2^n} + \frac{\left(\sum_{i=1}^q \mu_i\right)^2 + \sum_{i=1}^q \mu_i^2}{2^n} \\
&\leq \frac{1}{2^{n/2}} + \frac{2^{n/4}\left(2^{n/4} + 4 \cdot 2^{n/4}\right)}{2^n} + \frac{\left(2^{n/4} \cdot 2^{n/8}\right)^2 + 2^{n/4}\left(2^{n/8}\right)^2}{2^n} \\
&\leq \frac{1}{2^{n/2}} + \frac{5 \cdot 2^{n/2}}{2^n} + \frac{2^{3n/4} + 2^{n/2}}{2^n} \\
&\leq \frac{1}{2^{n/2}} + \frac{6 \cdot 2^{n/2} + 2^{3n/4}}{2^n} \\
&\leq \frac{7}{2^{n/2}} + \frac{1}{2^{n/4}}
\end{aligned}
$$

which is still negligible.

Our findings here show that SCP03 is well designed. Indeed, it does not only protect against a wide range of attacks, but it also remains secure even against powerful adversaries who attempt to deviate it from its secure behavior by asking many queries.

5.7 Security Proofs of Sf-nEtTw

5.7.1 Proof of Proposition 5.1

We start by providing three definitions that we will use throughout our proof.

MAC Function. Here, we just recall how a MAC scheme is related to its MAC function. Let $\mathcal{MA}[F] = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a MAC scheme based on the MAC function F. F takes as input a key K and a message m to output a tag τ. The tagging algorithm $\mathcal{T}_\mathsf{k}$ and the verification algorithm $\mathcal{V}_\mathsf{k}$ are defined as follows:

Tagging $\mathcal{T}_\mathsf{k}(m)$

1: $\tau \longleftarrow F_\mathsf{k}(m)$
2: **return** τ

Verification $\mathcal{V}_\mathsf{k}(m, \tau)$

1: **if** $F_\mathsf{k}(m) = \tau$ **then**
2: **return** 1
3: **else**
4: **return** 0
5: **end if**

Truncated MAC. Let $T : \{0,1\}^n \longrightarrow \{0,1\}^{n_T}$ be a transformation function. Let $\mathcal{MA}[F] = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a MAC scheme based on the MAC function F. We define the transformed MAC scheme $To\mathcal{MA} = (\mathcal{K}, To\mathcal{T}, To\mathcal{V})$ that uses ToF as its MAC function, where o denotes the composition operator. A *truncated MAC* is a transformed MAC in which $T(.)$ is the $\mathtt{MSB}_l(.)$ function that takes a message as input and returns the l most significant (i.e. left-most) bits.

Tweak Chaining MAC2 *($\mathcal{TC}$-$\widetilde{\mathcal{MA}}2$)*. Let $\widetilde{F}_\mathsf{k} : \{0,1\}^n \times \{0,1\}^* \longrightarrow \{0,1\}^n$ be a tweakable function and let $\mathcal{TC}$-$\widetilde{\mathcal{MA}}$ be its associated chaining scheme. We define $\mathcal{TC}$-$\widetilde{\mathcal{MA}}2$ as $\mathcal{TC}$-$\widetilde{\mathcal{MA}}$ except that $\mathcal{TC}$-$\widetilde{\mathcal{MA}}2$ operates on the entire tag returned by $\widetilde{F}(.,.)$ and not only on its half as in $\mathcal{TC}$-$\widetilde{\mathcal{MA}}$. Stated differently, $\mathcal{TC}$-$\widetilde{\mathcal{MA}}2$ is a MAC scheme in which the MAC function $F2$ is defined as follows:

MAC Function $F2_\mathsf{k}(m)$

$\tau \longleftarrow \widetilde{F}_\mathsf{k}(\mathtt{chaining}, m)$
$\mathtt{chaining} \longleftarrow \tau$
return τ

where $\mathtt{chaining}$ is a static variable that was initialized with $\mathbf{0}^n$.

Having thus presented the above definitions, we are now on a position to make our proof. Let $\widetilde{F}_\mathsf{k} : \{0,1\}^n \times \{0,1\}^* \longrightarrow \{0,1\}^n$ be a tweakable function and let $\mathcal{TC}$-$\widetilde{\mathcal{MA}}2[F2]$ be its associated tweak chaining MAC2 scheme. We notice that the $\mathcal{TC}$-$\widetilde{\mathcal{MA}} = (\widetilde{\mathcal{K}}, \widetilde{\mathcal{T}}, \widetilde{\mathcal{V}})$ scheme presented in Definition 5.8 can be seen as the truncated MAC of $\mathcal{TC}$-$\widetilde{\mathcal{MA}}2[F2]$, where $T(.) = \mathtt{MSB}_{n/2}(.)$. Thus, we denote the MAC function of $\mathcal{TC}$-$\widetilde{\mathcal{MA}}$ as $ToF2$.

Consider any polynomial-time SUF-CMA adversary $\mathcal{A}$ against $\mathcal{TC}$-$\widetilde{\mathcal{MA}}$. Recall that $\mathcal{A}$ can make two types of queries: tagging queries and verification queries. We suppose that $\mathcal{A}$ makes q tagging queries. We associate two adversaries to $\mathcal{A}$: an sPRF adversary $\mathcal{B}$ against the MAC function $F2$ (or equivalently against $\mathcal{TC}$-$\widetilde{\mathcal{MA}}2[F2]$), and an $\widetilde{\text{IND}}$-CPA distinguisher D against

the tweakable function $\widetilde{F}_{\mathsf{k}}$. Now, we state the following lemmas in which we define how the adversaries $\mathcal{A}$, $\mathcal{B}$ and D interact between each other and from which the proposition 5.1 follows directly.

Lemma 5.2. $\mathbf{Adv}^{\text{suf-cma}}_{\mathcal{A},\,\mathcal{TC}\text{-}\widetilde{\mathcal{MA}}}(k) = \mathbf{Adv}^{\text{sprf}}_{\mathcal{B},\,F2}(k) + 1/2^{n/2}$

Lemma 5.3. $\mathbf{Adv}^{\text{sprf}}_{\mathcal{B},\,F2}(k) \leq \mathbf{Adv}^{\widetilde{\text{ind-cpa}}}_{\mathsf{D},\,\widetilde{F}}(k) + \mathsf{Pr}[\mathsf{Col}_q] + \mathsf{Pr}[\mathsf{Col}]$

Lemma 5.4. $\mathsf{Pr}[\mathsf{Col}] \leq q^2/2^n$

Proof of Lemma 5.2: Recall that $\mathcal{B}$ has access to the oracle $\mathcal{O}$ and her goal is to distinguish whether $\mathcal{O}$ is the MAC function $F2$ or the stateful random oracle $\mathcal{R}_{\mathcal{S}}$. Recall also that the MAC function of $\mathcal{TC}\text{-}\widetilde{\mathcal{MA}}$ is $ToF2$. The algorithm $\mathcal{B}$ is described below:

Algorithm $\mathcal{B}^{\mathcal{O}}$

```
repeat
    if A queries (m) then
        τ ⟵ ToO(m)
        output τ to A
    end if
    if A queries (m, τ) then
        b ⟵ [τ = ToO(m)]
        output b to A
    end if
until A ends
if A forges then
    return 1
else
    return 0
end if
```

We can see that $\mathbf{B}$ perfectly simulates the answers to $\mathcal{A}$. In addition, $\mathcal{B}$ returns 1 (i.e. guesses that the oracle $\mathcal{O}$ is the MAC function $F2$) when $\mathcal{A}$ succeeds in forging a tag. Therefore, the following relation holds:

$$\mathsf{Pr}[\mathcal{A}^{ToO} \text{ forges}] = \mathsf{Pr}[\mathcal{B}^{\mathcal{O}} \Rightarrow 1] \tag{5.1}$$

where we use the equivalent notation in which we note that the adversary $\mathcal{A}$ has access to the MAC function as oracle instead of the tagging/verification oracles.

By definition of strong unforgeability of the MAC scheme $\mathcal{TC}\text{-}\widetilde{\mathcal{MA}}$ (see Definition 3.14), the advantage of $\mathcal{A}$ is defined by the probability of her success when she has access to the oracle $ToF2$. Therefore, we have:

$$\begin{aligned}
\mathbf{Adv}^{\text{suf-cma}}_{\mathcal{A},\,\mathcal{TC}\text{-}\widetilde{\mathcal{MA}}}(k) &= \mathsf{Pr}[\mathcal{A}^{ToF2} \text{ forges}] \\
&= \mathsf{Pr}[\mathcal{A}^{ToF2} \text{ forges}] + \big(\mathsf{Pr}[\mathcal{A}^{ToR_{\mathcal{S}}} \text{ forges}] - \mathsf{Pr}[\mathcal{A}^{ToR_{\mathcal{S}}} \text{ forges}]\big) \\
&= \big(\mathsf{Pr}[\mathcal{A}^{ToF2} \text{ forges}] - \mathsf{Pr}[\mathcal{A}^{ToR_{\mathcal{S}}} \text{ forges}]\big) + \mathsf{Pr}[\mathcal{A}^{ToR_{\mathcal{S}}} \text{ forges}] \\
&= \big(\mathsf{Pr}[\mathcal{B}^{F2} \Rightarrow 1] - \mathsf{Pr}[\mathcal{B}^{R_{\mathcal{S}}} \Rightarrow 1]\big) + \mathsf{Pr}[\mathcal{A}^{ToR_{\mathcal{S}}} \text{ forges}] \text{ (from 5.1)} \\
&= \mathbf{Adv}^{\text{sprf}}_{\mathcal{B},\,F2}(k) + \mathsf{Pr}[\mathcal{A}^{ToR_{\mathcal{S}}} \text{ forges}]
\end{aligned}$$

Now, we examine $\mathsf{Pr}[\mathcal{A}^{ToR_{\mathcal{S}}}$ forges], which is equal to the probability that $\mathcal{A}$ forges against a MAC scheme that has $ToR_{\mathcal{S}}$ as its MAC function. Recall that $R_{\mathcal{S}}$ is a random oracle. Let us suppose that (M, τ) is the forging query that $\mathcal{A}$ uses to break the scheme. Therefore, the following relations holds: $\tau = T\left(R_{\mathcal{S}}(m)\right)$. Thus, we have

$$\begin{aligned}\mathsf{Pr}[\mathcal{A}^{ToR_{\mathcal{S}}} \text{ forges}] &= \mathsf{Pr}[x \xleftarrow{R} \{0,1\}^n,\ T(x) = \tau] \\ &= \frac{1}{2^{n/2}} \qquad \big(\text{since } T(.) = \mathtt{MSB}_{n/2}(.)\big)\end{aligned}$$

which concludes our proof.

Proof of Lemma 5.3: Here, we consider any sPRF adversary $\mathcal{B}$ against $F2$ and we associate it to a particular $\widetilde{\text{IND}}$-CPA distinguisher D against the tweakable function $\widetilde{F}_{\mathsf{k}} : \{0,1\}^n \times \{0,1\}^* \longrightarrow \{0,1\}^n$. Recall that D makes q queries to the oracle $\mathcal{O}$ and her goal is to distinguish whether $\mathcal{O}$ is $\widetilde{F}_{\mathsf{k}}(.,.)$ or $\widetilde{\mathcal{R}}(.,.)$, where $\widetilde{\mathcal{R}}(.,.)$ is a function that, on input (T, m), returns n-bit random strings. Recall also that D is a tweak-respecting adversary (i.e. does not repeat tweak). We define the algorithm of D as follows:

Algorithm $\mathsf{D}^{\mathcal{O}}$

```
t ⟵ 0^n
S ⟵ {t}
repeat
    if B queries (m) then
        t ⟵ O(t, m)
        S ⟵ S ∪ {t}
        output t to B
        if S contains duplicate values then
            return 1
        end if
    end if
until B outputs b'
return b'
```

where S is a multiset in which values can repeat. We argue that when S does not contain the same value twice, D is perfectly simulating $\mathcal{B}$'s execution environment. This is true because $\widetilde{F}(.,.)$ is no distinguishable from the random oracle $\widetilde{\mathcal{R}}(.,.)$ only against tweak-respecting adversaries. We illustrate the importance of such a condition by an example. In our example, we take $\widetilde{\text{OMAC}}(T, m)$ $(= \text{OMAC}(T||m))$ as the tweakable function $\widetilde{F}(.,.)$. Now, we show that $\mathcal{B}$ can easily see under the simulation environment that she is not interacting with a random oracle. $\mathcal{B}$ knows that the initial tweak (i.e. state) is $\mathbf{0}^n$ and queries $m_1 = \mathbf{0}^n$ to receive τ_1 from her oracle. Then, let T be a tweak that repeats twice. For the first occurrence of T, $\mathcal{B}$ queries $m_2 = \mathbf{0}^n$ to receive τ_2 and for its second occurrence she queries $m_2 = \mathbf{0}^n \,||\, \tau_2 \,||\, \mathbf{0}^n$ to receive τ_3. It is easy to see that $\tau_1 = \tau_3$ when $\mathcal{O} = \widetilde{F}_{\mathsf{k}}(.,.)$ $(\neq \widetilde{\mathcal{R}}(.,.))$. Indeed, we have

$$\begin{aligned}\tau_1 &= E_{\mathsf{k}}\left(E_{\mathsf{k}}(\mathbf{0^n})\right) \\ \tau_2 &= E_{\mathsf{k}}\left(E_{\mathsf{k}}(T)\right) \\ \tau_3 &= E_{\mathsf{k}}\left(E_{\mathsf{k}}\left(\cancel{E_{\mathsf{k}}(E_{\mathsf{k}}(T))} \oplus \cancel{\tau_2}\right) \oplus \mathbf{0^n}\right)\end{aligned}$$

Thus, from D's algorithm, we can see that

$$\mathsf{Pr}[\mathsf{D}^{\widetilde{F}} \Rightarrow 1] = \mathsf{Pr}[\mathcal{B}^{F2} \Rightarrow 1] + \mathsf{Pr}[\mathsf{S}|\widetilde{F}]$$
$$\mathsf{Pr}[\mathsf{D}^{\widetilde{\mathcal{R}}} \Rightarrow 1] = \mathsf{Pr}[\mathcal{B}^{R_{\mathcal{S}}} \Rightarrow 1] + \mathsf{Pr}[\mathsf{S}|\widetilde{\mathcal{R}}]$$

where $\mathsf{Pr}[\mathsf{S}]$ is the probability that the multiset S contains duplicate values.

By using the two above relations, we get

$$\begin{aligned}\mathbf{Adv}^{\text{sprf}}_{\mathcal{B},F2}(k) &= \mathbf{Adv}^{\widetilde{\text{ind-cpa}}}_{\mathsf{D},\widetilde{F}}(k) + \overbrace{\mathsf{Pr}[\mathsf{S}|\widetilde{\mathcal{R}}]}^{\mathsf{Pr}[\mathsf{Col}]} - \overbrace{\mathsf{Pr}[\mathsf{S}|\widetilde{F}]}^{\mathsf{Pr}[\mathsf{Col}_q]} \\ &\leq \mathbf{Adv}^{\widetilde{\text{ind-cpa}}}_{\mathsf{D},\widetilde{F}}(k) + \mathsf{Pr}[\mathsf{Col}] + \mathsf{Pr}[\mathsf{Col}_q]\end{aligned}$$

where $\mathsf{Pr}[\mathsf{Col}_q]$ is the collision probability of the tweakable function $\widetilde{F}$ after q messages. Thus, our proof ends.

Proof of Lemma 5.4: Informally speaking, the lemma means that the set $\{x : x_0 = \mathbf{0}^n,\ x_i = \widetilde{\mathcal{R}}(x_0, .)\}$ has asymptotically negligible probability to include duplicate values. Recall that q is the number of $\mathcal{B}$'s queries. We start our proof by making induction on q. For all $q \geq 1$, we prove that

$$\mathsf{Pr}[\mathsf{Col}] = \frac{q(q+1)}{2^{n+1}} \tag{5.2}$$

Then, we conclude our proof by noticing that $q(q+1)/2^{n+1} \leq q^2/2^n$.

Base case. When $q = 1$, the right side of 5.2 is $1/2^n$. Now, let's look at the left side. After only one call, there are two elements in S: $\{\mathbf{0}^n, y\}$, where $y \longleftarrow \widetilde{\mathcal{R}}(\mathbf{0}^n, .)$. Thus, $\mathsf{Pr}[\mathsf{Col}] = \mathsf{Pr}[x \xleftarrow{R} \{0,1\}^n, x = \mathbf{0}^n]$, which is equal to $1/2^n$.

Induction step. Suppose that the equation 5.2 is true for $q = m - 1$. Here, x_i denotes the ith element of the multiset S. After $q = m$ calls, we have

$$\begin{aligned}\mathsf{Pr}[\mathsf{Col}] &= \overbrace{\mathsf{Pr}[\mathsf{Col} \text{ after } m-1 \text{ calls}]}^{\text{induction hypothesis}} + \mathsf{Pr}[x_m \in \mathsf{S}] \\ &= \frac{m(m-1)}{2^{n+1}} + \frac{m}{2^n} \\ &= \frac{m(m+1)}{2^{n+1}}\end{aligned}$$

Hence, the equation 5.2 holds for $q = m$, and the induction step is complete.

5.7.2 Proof of Theorem 5.1

Recall that $\mathcal{A}$ can make two types of queries: encryption queries and decryption queries. We denote $\mathcal{A}$'s i-th encryption query as M_i and the returned ciphertext as $C_i = \sigma_i || \tau_i$. We denote $\mathcal{A}$'s i-th decryption query as $C'_i = \sigma'_i || \tau'_i$ and the returned message as m_i. We associate to $\mathcal{A}$ an SUF-CMA forger $\mathcal{F}$ against $\mathcal{TC}$-$\widetilde{\mathcal{MA}}$. This association is similar to the one given in the Case 1 of Theorem 5.4: $\mathcal{F}$ generates a key $\mathsf{K1} \in \mathtt{Key}$ that she uses for the encryption/decryption algorithms of Sf-$n\mathcal{SE}$. We recall that the forger $\mathcal{F}$ has access to two oracles: a tagging oracle $\widetilde{\mathcal{T}}_{\mathsf{k2}}$ and a verification oracle $\widetilde{\mathcal{V}}_{\mathsf{k2}}$, where the key $\mathsf{K2}$ is independent from $\mathsf{K1}$. Below, we describe our trivial association.

1. When $\mathcal{A}$ makes an encryption query M, $\mathcal{F}$ outputs $\sigma \longleftarrow \mathsf{Enc}\,(n\mathcal{E}_{\mathsf{k1}}\text{-Sf}(M))$. Then, she queries σ to her tagging oracle $\widetilde{\mathcal{T}}_{\mathsf{k2}}$ and receives τ in response. Finally, she outputs $C = \sigma \,||\, \tau$ to $\mathcal{A}$.

2. When $\mathcal{A}$ makes a decryption query $C = \sigma \,||\, \tau$, the forger $\mathcal{F}$ queries τ to her verification oracle $\widetilde{\mathcal{V}}_{\mathsf{k2}}$ and receives a binary value b. If b is false, then $\mathcal{F}$ halts after outputting $\perp$. Otherwise, $\mathcal{F}$ computes $n\mathcal{D}_{\mathsf{k1}}\text{-Sf}\,(\mathsf{Dec}(\sigma))$ and outputs the result to $\mathcal{A}$.

3. When $\mathcal{A}$ wins in her INT-SFCTXT experiment, namely providing a new valid out-of-sync decryption query $C = \sigma \,||\, \tau$, then $\mathcal{F}$ stops and attempts to evaluate the pair (σ, τ) in order to see whether she succeeds in her forgery. The different cases are presented below in the proof of Lemma 5.5.

Now, suppose $\mathcal{A}$ has made q encryption queries and d decryption ones. Let j be the index of $\mathcal{A}$'s first out-of-sync decryption query. We only consider the first out-of-sync query because if it fails, the decryption algorithm will return $\perp$ and halt for all ensuing queries (see our discussion about the approach of halting state in Section 5.5.1). We define two events in case the $\mathcal{A}$'s j-th decryption query succeeds: (1) Col: $\exists i \leq q$ such that $\tau'_j = \tau_i$ and $i \neq j$; (2) Bad: $q \geq j$, $\tau'_j = \tau_j$ and $m_j = M_j$. We state the following lemmas from which Theorem 5.1 follows directly (using Proposition 5.1).

Lemma 5.5. $\mathbf{Adv}^{\text{int-sfctxt}}_{\mathcal{A},\,\text{Sf-nEtTw}}(k) \leq \mathbf{Adv}^{\text{suf-cma}}_{\mathcal{F},\,\mathcal{TC}\text{-}\widetilde{\mathcal{MA}}}(k) + \mathsf{Pr}[\mathsf{q\text{-}Col}] + \mathsf{Pr}[\mathsf{Bad}]$

Lemma 5.6. $\mathsf{Pr}[\mathsf{Bad}] = 0$

Proof of Lemma 5.5: As said previously, $\mathcal{A}$ made q encryption queries before her first out-of-sync query ($\mathcal{Q}$) which is the j-th decryption query ($C'_j = \sigma'_j \,||\, \tau'_j$). We define the following events.

E : $\mathcal{Q}$ correctly verifies
E_1 : E occurs and $\tau'_j \notin \{\tau_1, ..., \tau_q\}$
E_2 : E occurs and $\tau'_j \in \{\tau_1, ..., \tau_q\}$
$E_{2,1}$: E_2 occurs and either $q < j$ or $\tau'_j \neq \tau_j$
$E_{2,2}$: E_2 occurs and $q \geq j$ and $\tau'_j = \tau_j$
$E_{2,2,1}$: $E_{2,2}$ occurs and $m_j = M_j$
$E_{2,2,2}$: $E_{2,2}$ occurs and $m_j \neq M_j$

If $\mathcal{Q}$ fails, then $\mathcal{A}$ cannot win any more, since the decryption algorithm will return $\perp$ for any subsequent query. Therefore, $\mathbf{Adv}^{\text{int-sfctxt}}_{\mathcal{A},\,\text{Sf-nEtTw}}(k) = \mathsf{Pr}[E]$. Considering the different events, we have $\mathsf{Pr}[E] = \mathsf{Pr}[E_1 \vee E_{2,2,2}] + \mathsf{Pr}[E_{2,1}] + \mathsf{Pr}[E_{2,2,1}]$.

Now, we study the probabilities of these events. We can see that $E_{2,1}$ corresponds to the event Col, since it implies that τ'_j has already been produced before and that was not during the j-th encryption query (this includes the fact that $\mathcal{A}$ might not have made j encryption queries yet). Concerning $E_{2,2,1}$, it is easy to see that it satisfies the definition of the Bad event. Consequently, we have

$$\mathbf{Adv}^{\text{int-sfctxt}}_{\mathcal{A},\,\text{Sf-nEtTw}}(k) = \mathsf{Pr}[E_1 \vee E_{2,2,2}] + \mathsf{Pr}[\mathsf{Col}] + \mathsf{Pr}[\mathsf{Bad}]$$

We conclude the proof by examining $\mathsf{Pr}[E_1 \vee E_{2,2,2}]$ and $\mathsf{Pr}[\mathsf{Col}]$.

$\mathsf{Pr}[E_1 \vee E_{2,2,2}]$. We notice that when the event $E_1 \vee E_{2,2,2}$ occurs, (i.e. the j-th decryption oracle $C'_j = \sigma'_j \,||\, \tau'_j$ does not return $\perp$), then the forger $\mathcal{F}$ succeeds in finding an SUF-CMA forgery against $\mathcal{TC}\text{-}\widetilde{\mathcal{MA}}$, since the two events ensure that the pair (m_j, τ'_j) was never produced before by the oracle $\widetilde{\mathcal{T}}_{\mathsf{k2}}$.

Indeed, the event E_1 implies that τ'_j has never been queried to $\widetilde{\mathcal{T}}_{\mathsf{k2}}$, while the event $E_{2,2,2}$ implies that the tag τ'_j has never been obtained from querying the oracle $\widetilde{\mathcal{T}}_{\mathsf{k2}}$ with σ'_j as input. This is because for any state i, the following implication is asymptotically true (i.e. $n\mathcal{E}$-Sf(.) is injective):

$$M_i \neq M'_i \Longrightarrow n\mathcal{E}_{\mathsf{k1}}\text{-Sf}(M_i) \neq n\mathcal{E}_{\mathsf{k1}}\text{-Sf}(M'_i)$$

Therefore, τ'_j $(= \tau_j)$ was computed for $\sigma_j = n\mathcal{E}_{\mathsf{k1}}\text{-Sf}(M_j)$ which is different from σ'_j (i.e. $\sigma_j \neq \sigma'_j$), since $\sigma'_j = n\mathcal{E}_{\mathsf{k1}}\text{-Sf}(m_j)$ and $M_j \neq m_j$.

Thus, we have

$$\mathsf{Pr}[E_1 \vee E_{2,2,2}] = \mathsf{Pr}[\mathcal{F}forges] = \mathbf{Adv}^{\text{suf-cma}}_{\mathcal{F},\mathcal{TC}\text{-}\widetilde{\mathcal{MA}}}(k)$$

Pr[Col]. As previously pointed out, $\mathsf{Pr}[\mathsf{Col}] = \mathsf{Pr}[\exists i \neq j$ such that $\tau'_j = \tau_i]$. This means that for two different states the following equality holds:

$$\widetilde{\mathcal{T}}_{\mathsf{k2}}(\mathsf{Enc}(n\mathcal{E}_{\mathsf{k1}}\text{-Sf}(m_j))) = \widetilde{\mathcal{T}}_{\mathsf{k2}}(\mathsf{Enc}(n\mathcal{E}_{\mathsf{k1}}\text{-Sf}(M_i)))$$

The above relation supposes that the adversary should find a collision against $\mathcal{TC}\text{-}\widetilde{\mathcal{MA}}$ after q invocations to the $\widetilde{\mathcal{T}}_{\mathsf{k2}}(.)$ oracle, which corresponds to find a state $i(\neq j)$ such that the related MAC tag is equal to the one computed for the state j. By looking at the construction of $\mathcal{TC}\text{-}\widetilde{\mathcal{MA}}$ in Definition 5.8, we find that $\mathsf{Pr}[\mathsf{Col}]$ is equivalent to the probability of encountering a collision against the underlying tweakable function $\widetilde{F}_{\mathsf{k2}}(.,.)$ after q messages. Stated differently, we have

$$\mathsf{Pr}[\mathsf{Col}] = \mathsf{Pr}[\mathsf{q\text{-}Col}]$$

Proof of Lemma 5.6: The event $E_{2,2,1}$ includes all the following events: (1) $q \geq j$; (2) the decryption query $C'_j = \sigma'_j || \tau'_j$ is out-of-sync, hence $C_j \neq C'_j$; (3) $\sigma'_j \neq \sigma_j$, since $\tau'_j = \tau_j$; and (4) $m_j = M_j$. We notice that the events 3 and 4 are contradictory, and therefore $\mathsf{Pr}[\mathsf{Bad}] = 0$. Indeed, recall that $n\mathcal{SE} = (\mathcal{K}, n\mathcal{E}, n\mathcal{D})$ encrypts messages using a deterministic algorithm. Therefore, for a fixed nonce n, $n\mathcal{E}_{\mathsf{k1}}(n, m_1) \neq n\mathcal{E}_{\mathsf{k1}}(n, m_2)$ implies that $m_1 \neq m_2$. Also, we notice that the encryption and the decryption states were in-sync prior to the j-th decryption query. This means that the associated Sf-$n\mathcal{SE} = (\mathcal{K}\text{-Sf}, n\mathcal{E}\text{-Sf}, n\mathcal{D}\text{-Sf})$ has called $n\mathcal{SE}$ algorithms with the same nonce for each state. Thus, the event 3 entails $\sigma'_j = n\mathcal{E}_{\mathsf{k1}}\text{-Sf}(m_j) \neq n\mathcal{E}_{\mathsf{k1}}\text{-Sf}(M_j) = \sigma_j$, which implies $m_j \neq M_j$. This concludes our proof, since the event 4 is $m_j = M_j$.

5.8 Discussion

An important aspect of any cryptanalysis is what it implies in practice. Our study reveals interesting facts about the family of SCP. In particular, two protocols are concerned: SCP02 and SCP03. Here, we discuss our findings.

While discussing our results, we are aware that provable security is not a silver bullet for security, as authors of [DPW11] notice that several cryptographic schemes have been proved secure and then broken some years later. We argue that this fact does not nullify the interest of such a powerful security tool. Indeed, despite being imperfect, provable security has greatly helped ruling out a large class of attacks in security protocols. In addition, although its findings should not be taken as absolute, they constitute a general direction that aims at designing better cryptographic schemes.

The vulnerable, yet popular, SCP02. In section 5.4, we see that, unlike extensive evaluation, provable security for certified products provides a strong guarantee of security without promoting complexity. Indeed, we demonstrated a theoretical attack against the protocol SCP02.

In addition, we showed how some technical details about SCP02 make the attacker likely to succeed in the context of smart cards. Surprisingly, the presented attack arises from a fundamental design flaw in SCP02, which is the use of CBC mode with no IV.

It is not clear that why the SCP02 designers made such a choice. However, we might suspect that the reasons behind this are twofold. First, when the first variant of SCP02 was published in 2000, cryptographic results about using CBC mode with stateful nonce-based IVs were not well-established yet. Second, designers chose not to use random IVs in order to reduce the overhead of SCP02. Indeed, a random IV must be appended to the sent ciphertext, thereby increasing the communication overhead with the smart card. In addition, the implementations of CBC mode in smart cards have been optimized to pre-generate some objects during the initialization of the cipher object. The problem is that choosing the IV is uniquely done together with the choice of the encryption key during initialization. Therefore, constantly modifying the IV implies constant initialization of the cipher object that can no longer performs its optimization in advance. Thus, we argue that the real challenge of SCP02 was to achieve good performance in a limited environment, like a smart card, and still ensuring security. In the complex GP card specifications, the tiny detail of 'just keep using the same IV' might have passed unnoticed, especially that to the best of our knowledge, no formal analysis of SCP02 has been performed before.

Furthermore, identifying such a well-known vulnerability tells us something: smart cards industry has difficulty in catching up with the advances on cryptography. Fianlized in 2003, SCP02 keeps existing, while other protocols have continuously been updated. Ironically, the stringent requirements of smart cards about security are both its strongest and weakest point: they do not make this technology only secure and trustworthy, but also so slow to improve. We illustrate by three examples. First, EMV [EMV], which is the actual standard of payment, still mandates the use of Triple DES with two independent keys instead of using AES (see Section 5.7 in the EMV Card Personalization Specification [EMV07]). Second, numerous card manufacturers continue relying on SCP02, although SCP03 was published in 2009. For instance, NXP instructs the support of SCP02 and makes it optional for SCP03 for all its JCOP products that are certified EAL5+ [Gmb14]. Third, the SCP family (i.e. SCP02 and SCP03) still requires encrypting data using the CBC mode. As a matter of fact, Mitchell in [Mit05b] (and more recently Rogaway in [Rog11]) promotes abandoning CBC for future designs.

The powerful SCP03. Introduced as an amendment in 2009, we have analyzed SCP03 in sections 5.5 and 5.6 and have found that it provably satisfies strong security notions. Of a particular interest, we proved that SCP03 resists against the algorithm substitution attacks (ASAs) that could lead to secret mass surveillance [BPR14]. This result is significant, as it increases the trust in the closed industry of smart cards. The advantages offered by SCP03 are clear: it is provably secure and it is being gradually implemented by card manufacturers. It is true that the added security comes with additional cost: maintaining a 2-byte counter (i.e state) per session as well as one more block cipher invocation per message (recall that the counter is encrypted in order to be used as an IV). However, modern smart cards include a dedicated cryptographic co-processor, hence the incurred overhead is very small.

Standards are particularly susceptible to significant modification. Therefore, we feel that the recently created GP 'Crypto Sub-Task Force' [Glo16a] may have a hard time justifying to wholly reconsider the design of the SCP family. Therefore, we advocate the deprecation of SCP02 as soon as possible and the switch over to SCP03 that should be included in the main specification instead of being an amendment. Our goal is to provide enough information to the GP community so that the Crypto Sub-Task Force can take an informed decision when deciding how to fix the current problems with SCP02. At this point, a quote from [BKN02] seems appropriate: "*in the modern era of strong cryptography, it would seem counterintuitive to voluntarily use a protocol with low security when it is possible to fix the security (...) at low cost*".

Part III

Trust on Smart Complex Systems

Chapter 6

Towards the Dual-Execution-Environment Approach

"It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts."

– Conan Doyle, Sherlock Holmes, A Scandal in Bohemia

Contents

6.1 Introduction **100**
6.1.1 Previous Work 100
6.1.2 Overview of our Comparative Framework 102
6.2 Background **102**
6.2.1 Trusted Model 102
6.2.2 MILS 103
6.3 Related Work **104**
6.4 The Dual-EE Approach **104**
6.4.1 From Dual-EE to Multi-EE 105
6.4.2 Classification of Dual-EE Solutions 106
6.5 Comparison Methodology **107**
6.5.1 Functional Criteria 107
6.5.2 Security Criteria 108
6.5.3 Deployability Criteria 108
6.6 Comparative Evaluation **109**
6.6.1 External Hardware Module: Secure Element 109
6.6.2 Bare-Metal Hypervisor: KVM/ARM 109
6.6.3 Special Processor Extensions: TrustZone 110
6.7 Discussion **111**
6.8 Conclusions **112**

The dual-execution-environment approach (dual-EE) is a trusted model that was defined to allow mobile smart devices to guarantee tamper-resistant execution for highly sensitive applications. Although various solutions implementing dual-EE have been proposed in the literature, this model has not been formalized yet. In this chapter, we revisit the dual-EE approach and propose

a theoretical framework to systematize the design of dual-EE solutions regarding well-established primitives defined in the Multiple Independent Levels of Security (MILS) architecture. We provide a general classification of the different dual-EE proposals based on their isolation properties. We introduce a comparative framework allowing dual-EE solutions to be evaluated across a common set of criteria. The relevance of our framework is examined by applying it on three technologies, each one represents one category in our classification. Results are consistent and explain some hidden and unexpected properties of each technology. For instance, we find that baremetal hypervisors are ill-adapted to provide high assurance security even though they might improve the overall security level of the system. The findings of this chapter were published in [SAB15a] that was presented at the 2015 *IFIP Information Security & Privacy Conference (IFIP SEC 2015).*

6.1 Introduction

The wide use of modern mobile devices spurs service providers to propose access to their services via smart devices. The growing number of attacks against such devices puts mobile applications under potential security risks [FBL+15, LPMS13]. Thus, smart devices are not mature yet for supporting services requiring trusted platforms with proved security. Examples include enterprise applications and NFC-based payment solutions. Indeed, the adoption of mobile devices in sensitive business environments has been hindered by the lack of appropriate level of security.

Sensitive-service providers require that their applications run on tamper-resistant execution environment. Such an environment should at least guarantee the following three properties [SLS+05]:

1. *authenticity*: the code under execution should not have been changed;
2. *integrity*: during execution, runtime states (e.g. CPU registers, memory and sensitive I/O) should not have been tampered with;
3. *privacy*: code, data and runtime states should not be observable by unauthorized applications or even underlying OS that might have been compromised.

As we have seen in chapter 2 (see section 2.2), the default protection mechanisms of smart devices are insufficient to provide tamper-resistant environment. We recall that this is due to the fact that these protection mechanisms are mainly based on the operating system, and thus as long as the operating system has not been compromised, sensitive applications are considered as protected. Unfortunately, despite continued efforts to improve the security of operating systems of smart devices [STCT15, PS10], they are still essentially untrustworthy for two reasons. First, they are complex and often developed using unsafe languages. Therefore, they are inherently error prone because design flaws and implementation bugs are unavoidable. Indeed, authors in [CZ11] have empirically shown that there is a strong correlation between security vulnerabilities and complexity. A more thorough study about complexity and its attributes is presented in [FP98]. Second, they allow poor isolation among applications. Indeed, a process with the root privilege can easily access private data and tamper with the execution of other processes. Thus, applications with diverse security requirements cannot be run concurrently, since the security level of the system is simply reduced to that of its most vulnerable application.

6.1.1 Previous Work

To remedy this situation, there have been numerous efforts aimed at providing tamper-resistant execution environments. Generally, those efforts can be classified into four categories.

- **Processor Enhancements Based Approach.** This approach allows sensitive applications to run on untrustworthy operating system. Solutions adopting this approach include

AEGIS [SCG+03], XOMOS [LTH03], Bastion [CL10] and SecureME [CRSP11]. These solutions usually use trusted system software (often a tiny kernel) along with a specialized hardware to achieve various security and privacy requirements. In particular, they support fine-grained memory protection to defend specific applications. Indeed, they are able to protect applications against powerful attackers that can snoop buses and modify memory by encrypting data and code before it is sent to memory. Despite the strength of its guaranteed security, this approach is not practical for current commodity mobile devices, since it requires non-trivial hardware modifications to the core processor architecture.

- **Architectural Enhancements Based Approach.** This approach uses sandboxes to isolate security sensitive code from the OS kernel. A sandbox can be used to host kernel monitoring and protection tools. Solutions adopting this approach include Flicker [MPP+08], Inktag [HKD+13] and Virtual Ghost [CDA14]. However, these solutions use specially tailored operating systems to provide the isolation. Hence, they can only have very limited success due to their high cost and incompatibility to existing applications.

- **Micro-Kernel Based Approach.** This approach tries to reduce the software trusted computer base (TCB) by running a small code in the privileged mode. Solutions adopting this approach include SeL4 [KEH+09], L4 Android [LLL+11] and the HACMS program for military systems [Fis15]. These solutions often aim at providing formally verified micro-kernels to have high assurance of security. In spite of their apparent attractiveness, these solutions impose strict requirements on the micro-kernel design, thereby requiring significant efforts for their implementations. We also note that the size of micro-kernels that can be formally verified is around 10K lines of code (LOC) [EH13], whereas modern operating systems for smartphones greatly exceed this size. Therefore, this approach is only applicable in very few cases in practice, since it necessitates significant modifications to port existing applications.

- **Dual-Execution-Environment Approach (Dual-EE).** This approach splits the system into two strongly isolated subsystems, one of which is used to host a specialized OS with restricted functionalities. It relies on the specialized OS to provide tamper-resistant capabilities, while the other subsystem runs the main OS that executes most of the applications. Applications that demand tamper-resistant protection run only on the specialized trustworthy OS. Solutions adopting this approach include Nizza [HHF+05], Fides [SP12] and TLR [SRSW14].

Compared to other approaches, the dual-EE is considered as a promising approach intended for practical use [GPC+03]. The literature is full of proposals [GPC+03, WFM+07, KEAR09, GPHB11]. However, proposals differ substantially from each other in their design objectives. Some address very specific environments, while others silently seek generic solutions that fit all environments. Too often, authors claim the superiority of their solutions and their assertion is based on self-defined criteria To make progress, we believe that knowledge regarding the dual-EE approach must be systematized. There is a need to provide a theoretical framework which defines how best to evaluate dual-EE proposals.

In this chapter, we analyze the dual-EE approach in the context of the trusted computing domain and the MILS architecture. We propose a standard benchmark and framework allowing dual-EE solutions to be rated across a common, broad spectrum of criteria. Our work provides insights which prove useful in designing more efficient dual-EE schemes (see chapters 7 and 8). To the best of our knowledge, this is the first comparative evaluation of the dual-EE solutions available on smartphones or more generally on mobile smart devices. Moreover, we believe that our comparative framework is extendable and sufficiently general to be used to evaluate more fine-grained classifications.

6.1.2 Overview of our Comparative Framework

The basic idea of our framework can be summarized in three axes as follows.

1. We construct a compact security model of the dual-EE approach using the *separation kernel* and the Multiple Independent Levels of Security models that provide a relevant abstraction level, thereby contributing to a deeper understanding of the dual-EE approach. We reinterpret well-known security technologies, such as UICC card and TrustZone, in the light of this model.

2. We provide specific criteria to evaluate the dual-EE solutions. Our criteria are divided into three categories: (1) functional criteria: schemes are evaluated whether they implement all the requirements of a tamper-resistant environment; (2) security criteria: the properties of the separation kernel layer of the scheme are analyzed; and (3) deployability criteria: schemes are evaluated whether they could be easily deployed in a real context.

3. Finally, we provide a classification of the different dual-EE solutions. Nevertheless, our goal is not to provide a comprehensive survey, but to show the relevance and the interest of our abstraction by providing a general classification on which our comparative framework could be applied.

6.2 Background

In order to better understand this chapter, readers are required to be familiar with some background information:

- Smartphone security and its security challenges (section 2.2);
- Secure Element (section 2.4.2);
- ARM TrustZone (section 2.4.3);
- Trusted Model (described below);
- The architecture of Multiple Independent Levels of Security (described below).

6.2.1 Trusted Model

Building a secure system has traditionally been a cat and mouse game. No sooner are new security mechanisms integrated into systems than hackers find how to bypass them. Research on trusted computing aims to replace this endless game with a methodical process. Success in achieving a high degree of security in a system depends on the degree of care put into designing and implementing its security mechanisms. However, the experience shows that security vulnerabilities still exist even after carefully applying the best software engineering practices.

The domain of trusted computing provides the abstract concepts as well as the theoretical base on which ideal secure systems are built [Gas88]. It introduces various security models, called **trusted models**. The purpose of a trusted model is to precisely express the security requirements of a system. Each trusted model defines a set of security objectives, a threat model, and security requirements to be satisfied by the component that enforces the security policy.

A trusted model should leave no loophole in characterizing the security mechanisms of a given system. Indeed, it is defined by several properties: (1) it is precise and unambiguous; (2) it is abstract and generic; and (3) it is simple, and therefore easy to be understood. For high security systems, the property of unambiguity is satisfied by writing the trusted model in a formal mathematical notation. However, it is unfair to imply that the only good way to specify a trusted

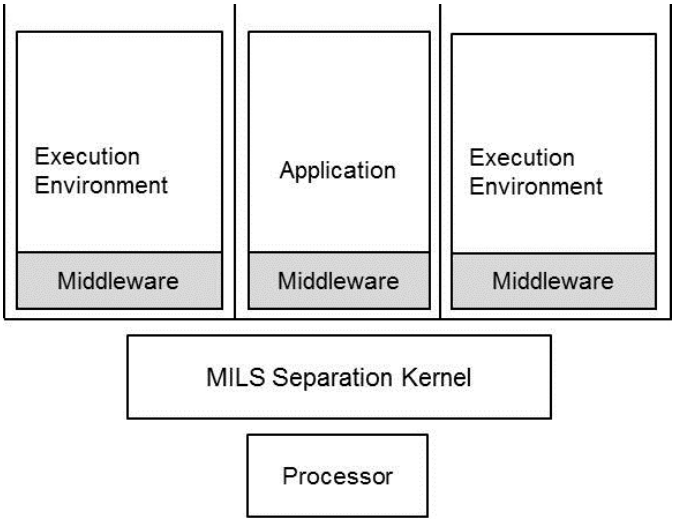


Figure 6.1: An Overview of the MILS Architecture

model is to use a mathematical model. Informal trusted models constitute an interesting balance between simplicity and strong security.

In this chapter, we focus solely on the 'separation kernel' trusted model introduced by John Rushby [Rus81], or more precisely, on MILS [VLB+05].

6.2.2 MILS

MILS stands for Multiple Independent Levels of Security. This architecture was developed in order to resolve the difficulty to evaluate the assurance level of the widely deployed trusted model 'reference monitor' [Lam74]. The 'reference monitor' model, originally defined to improve the security of a system by isolating the security layer in one component, is continually growing, and thus steadily falls short to achieve its primary goal.

In this context, MILS was defined to reduce the growing complexity of full-fledged systems by adopting a divide-and-conquer approach. Indeed, it separates a complex system that includes various modules requiring different levels of security into smaller, hence verifiable components. Thus, instead of evaluating the whole complex system, these small components are individually evaluated. An abstract view of the MILS architecture is depicted in Figure 6.1.

The MILS approach to security advocates 'component decomposition' as the first step in secure system design. The decomposition is logical, or virtual, in a way that it is unconcerned by the concrete realization of such separation. Often sharing the same underlying system, the isolated components might share some of the physical components, which introduces the possibility of of interference between logically components. Interference can result in propagation of security flaws. MILS addresses these concerns by requiring that shared components implement *separation* which guarantees that sharing of resources is undetectable. Thus, MILS is characterized by two steps: a first policy level that decomposes the system into distinct components, and a second resource-sharing level that defines how physical resources are allocated to these components. This approach differs from other security architectures (e.g., security kernel [AGS83]), where there is only one level of security that is managed by one component.

The primary component of MILS is the separation kernel layer (SK). This layer is responsible for creating a set of isolated functional units called *partitions*. Processes, including applications and execution environments, running in different partitions cannot communicate unless explicitly permitted by the SK. All communication between partitions is monitored by the SK layer. MILS is based on separation technology and secure inter-partition communication.

In order to work properly, the SK layer must satisfy several requirements. The SK should be designed so that it cannot be modified or disabled by rogue partitions. In addition, all inter-partition communication requests must go through it. Furthermore, it must be well-structured and small enough, so that its correctness can be validated. In other words, the SK must be (1) tamper-proof, (2) always invoked, and (3) evaluable. These properties correspond respectively to the three principles: isolation, completeness and verifiability.

6.3 Related Work

The two main research directions that our model targets is trustworthy execution and trusted computing in mobile devices. Extensive discussion of trusted computing solutions for mobile devices is found in [AEK+14]. Authors in [VOZ+12] evaluate existing hardware security features available on mobile devices for creating tamper-resistant execution. However, these surveys fail to identify dual-EE as a promising model that brings trusted computing for smartphones. Indeed, they just review various technologies without providing a deep insight. The absence of an appropriate level of abstraction makes the list of the described technologies keep being updated in a confusing way [MFAM16, AM16, MAM15].

Earlier works focus solely on a particular dual-EE technology discussing the advantages that it presents compared to other existing technologies. For instance, the case of TrustZone is presented in [WFM+07] and that of bare-metal hypervisors is presented in [GPHB11]. Too often, authors assert the superiority of their solution without explicitly stating their evaluation criteria. As such, consensus is unlikely as objective comparison between different solutions is not possible.

The closest work to ours is [EST14], both of which propose a standard benchmark and framework allowing dual-EE solutions to be evaluated across a common set of criteria. Authors in [EST14] construct their comparative framework on security functions which they define to cover the security risks for enterprise mobile applications. On the other hand, we construct our comparative framework on MILS, a well-known trusted model. The main advantage of using MILS is to provide a deeper comprehension of many hidden properties of the dual-EE approach. In addition, some may argue the impartiality of any framework built on self-cooked criteria, while the relevance and the objectivity of our criteria are guaranteed, since they are based on a thoroughly defined trusted model.

The main interest of our work is to systematize the study of architectures built upon the dual-EE approach. Our generic model is able to precisely characterize new solutions that are not described in this chapter. Indeed, Lesjak et al. have shown the relevance of our comparative framework by examining two different dual-EE solutions [LHW15]. Unlike our study, they rely on prototype implementations to provide their evaluation. Nevertheless, their findings are consistent to ours: isolation based on special processor extension promises greater flexibility, whereas isolation based on external hardware module provides stronger protection. Thus, our framework could allow system developers to rapidly design hybrid architectures in order to combine the advantages of the different used technologies.

6.4 The Dual-EE Approach

As said previously, there is an increasing need to use mobile devices for applications requiring high security levels, such as enterprise and payment applications. However, the openness and complexity of such devices impose fundamental limitations on their security. The dual-EE approach attempts to resolve these limitations by not only providing trust and high-assurance security, but also keeping the rich model of smartphones. It brings the best properties of open and trusted systems to smartphones without any compromise between security and usability. The dual-EE approach can be seen as a particular case of the MILS architecture, where the separation kernel

creates two partitions only. Indeed, it partitions the system into two execution environments running side-by-side: general-purpose execution environment (GPEE), and secure execution environment (SEE). The GPEE runs the legacy, complex operating system, while the SEE runs a special secure OS with a selection of applications specifically tailored for it. The SEE is designed to be trustworthy to provide tamper-resistant capabilities. Figure 6.2 depicts the representation of the dual-EE approach in the MILS architecture.

Secure isolation is essential for the dual-EE approach. Generally, the security of a system is reduced to that of its most vulnerable component. In dual-EE, the security level is, by definition, supposed to be that of the GPEE. However, the two execution environments are strongly isolated so that any attack on the GPEE does not impact the SEE. In this chapter, we consider MILS as the abstract trusted model of the dual-EE approach in which only two partitions exist and the strong isolation is guaranteed by the SK layer. The main advantage of this representation is to use MILS properties as primitives to better understand and thoroughly analyze the dual-EE approach. For instance, MILS defines a set of design principles for the SK layer. These principles provide an abstract model to define the isolation properties required between the two execution environments. We discuss these principles in the next section.

6.4.1 From Dual-EE to Multi-EE

The SK layer, as it was designed by Rushby [Rus81], can run an arbitrary number of execution environments. Thus, the multi-EE approach can be defined similarly to the dual-EE one. Indeed, the multi-EE approach allows smartphones to execute different sensitive applications with different security policy, while the dual-EE approach is more limited, since it executes all the sensitive applications inside the same SEE that defines the same policy for all sensitive applications.

Intuitively, we might think that dual-EE is nothing but a special case of the multi-EE approach. However, we prove by induction that the opposite is true: any multi-EE architecture can be reduced to an equivalent dual-EE architecture. It is important to note that the given dual-EE architecture is only equivalent in terms of security properties. No other consideration is regarded in our reduction for two reasons. First, the use-case of running multiple environments inside a mobile device is not common. Indeed, Shuja et al. in their study [SGB+16] show that most of the mobile virtualization solutions ensuring security were designed to run no more than two virtual machines (i.e. operating systems). Second, virtualization solutions are not known for their flexibility or high performance. In fact, they perform better if their virtualization capabilities of running any number of OS are restricted. Below, we present our findings in more detail.

Theorem 6.1. [*All Multi-EE Architecture Can Be Reduced to a Dual-EE Architecture*]. Let $\mathcal{S}$ be a system in which the security is based on a separation kernel. If the requirements of all secure applications are equal, then all multi-EE architecture can be reduced to a finite number of embedded dual-EE architectures. The equivalence is valid only in terms of security and functionality.

Proof. We perform our induction proof on $\mathcal{N}$, the number of execution environments running inside multi-EE.

Base case. When $\mathcal{N} = 2$, the equivalence between multi-EE and dual-EE is trivial.

Induction step. suppose that Theorem 6.1 holds for $\mathcal{N} = m$. This implies that any multi-EE architecture running m execution environments can be reduced to an equivalent dual-EE one. Now, we check if it holds for $\mathcal{N} = m+1$. For the sake of simplicity, and without loss of generality, we only take the setting when $m = 2$. Extending our proof to the general case is easy.

Let $\mathcal{M}$ be a multi-EE architecture running 3 $(= m + 1)$ execution environments. Let $\mathcal{D}$ be an architecture following the dual-EE approach. By definition, $\mathcal{D}$ runs two environments. Now,

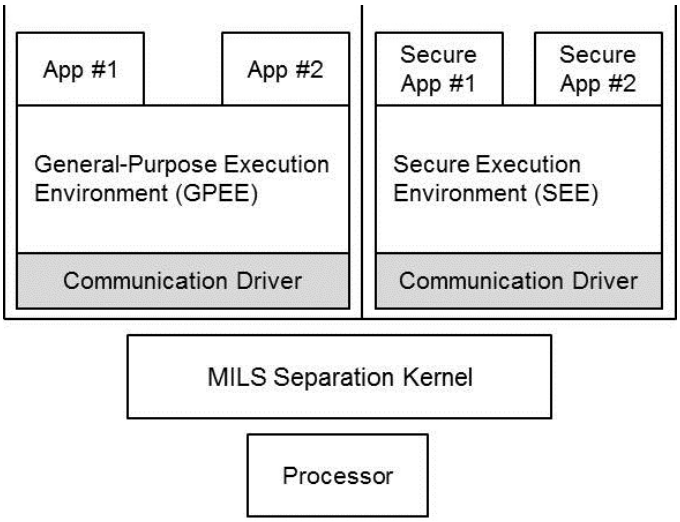


Figure 6.2: Representation of the Dual-EE Approach in the MILS Abstraction

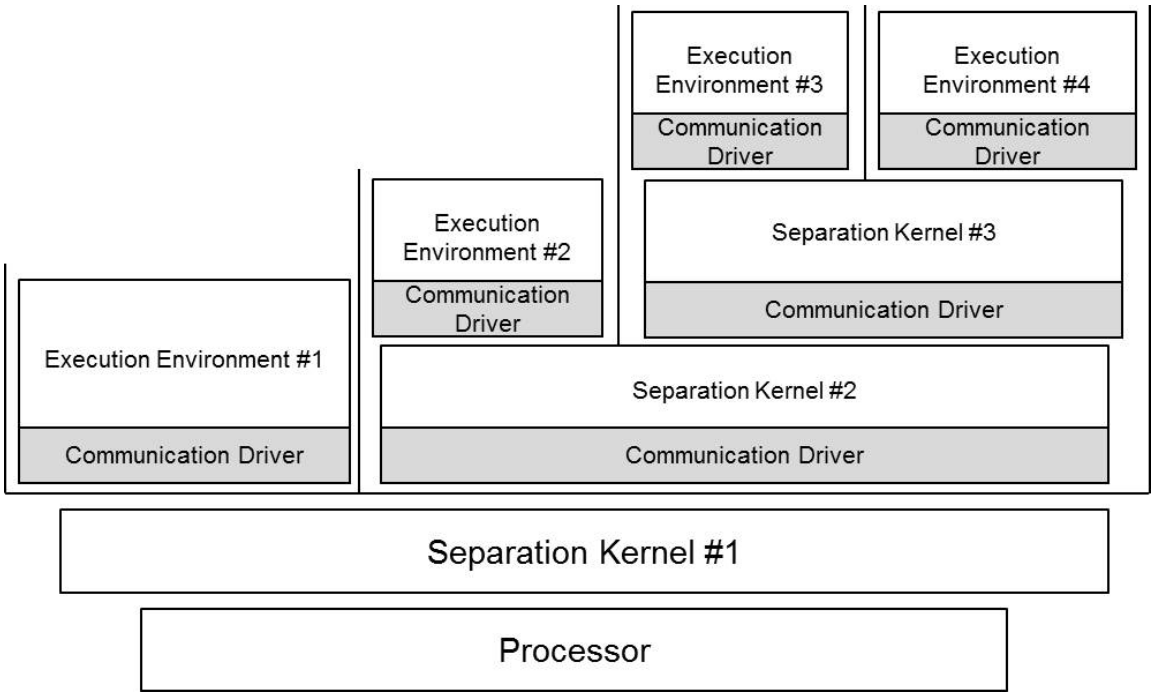


Figure 6.3: Proof by Picture When Multi-EE Runs Four Execution Environments

let us duplicate the $\mathcal{D}$ design inside $\mathcal{D}$ itself. This means that the SEE is replaced by a dual-EE SK that can run two other execution environments. Thus, the constructed architecture, denoted $\mathcal{D}'$, in overall executes three environments. Consequently, $\mathcal{M}$ can be equivalently designed as $\mathcal{D}'$ using only two embedded dual-EE architectures. Figure 6.3 depicts the case when $m = 4$. We notice that the equivalent architecture contains 3 $(= m - 1)$ SK layers. □

6.4.2 Classification of Dual-EE Solutions

According to their isolation technology, we classify the dual-EE solutions into three categories:

1. **Isolation based on external hardware module:** this category consists in introducing an additional secure coprocessor or integrated circuit to smartphones. A secure coprocessor

is a hardware module containing CPU, bootstrap ROM, and secure non-volatile memory. This hardware module is physically shielded from illegal access, and the I/O interface to the module is the only way to access its internal states. Hardware modules cannot only store cryptographic keys without risk of release, but also they can perform arbitrary computations using their CPU. In dual-EE, the SEE runs inside the secure coprocessor. Tamper-resistant execution is guaranteed, since the GPEE and the SEE run on physically two separated memories. Popular examples are Secure Element and the baseband processor;

2. **Isolation based on bare-metal hypervisor:** this category consists in executing a hypervisor in the most privilege mode of the processor. A *hypervisor* is a software layer that implements the same instruction-set architecture as the hardware on which it is executed. Thus, it allows multiple operating systems to coexist on the same hardware. Full-virtualization is not possible on ARM processors, which represents 85% of the market of modern mobile devices [ARM15b], since ARM is not a virtualizable architecture [DN14]. To remedy this problem, ARM introduced hardware virtualization support with the ARMv7 architecture. However, the use of hardware-supported virtualization on ARM is still limited [SGB+16]. Instead, the para-virtualization approach is prevalent and a myriad of solutions exists [PKR+13, DLC+12], since it is more flexible and does not monopolize hardware extensions. In para-virtualization, the OS needs to be modified in order to run on the underlying hypervisor. In dual-EE, the hypervisor plays the role of the SK layer, and the number of virtual machines is limited to two;

3. **Isolation based on special processor extensions:** this category consists in enhancing general-purpose processors with new hardware extensions. These newly-introduced extensions allow the execution of secure code within a potentially compromised OS. The most prevalent secure extensions targeting smartphones is ARM TrustZone (see section 2.4.3 in chapter 2). In this chapter, we only consider ARM TrustZone because it is the most deployed security extensions in practice [ARM15b]. We recall that a processor with TrustZone extensions provides a special form of virtualization. It enables two virtual processors with two security domains: the "secure" zone and the "normal" zone. In dual-EE, the GPEE resides in the normal zone and the SEE resides in the secure zone. The isolation of both zones or "worlds" is implemented by a complex mechanism using hardware controllers, a configuration bit and the new execution mode called *monitor mode*.

6.5 Comparison Methodology

In order to evaluate dual-EE solutions, we define three categories of criteria: functional, security and deployability.

6.5.1 Functional Criteria

Schemes are evaluated whether they implement all the requirements of a tamper-resistant environment. The SEE should provide the following features [Gra06]:

- *Protected Execution.* The execution of secure applications should be protected from any interference caused by malicious software. In addition, runtime states of the SEE should be protected from being observed or tampered with.
- *Sealed Storage.* The integrity, secrecy and freshness of secure applications' content should be protected. Content includes code as well as data.
- *Protected Input.* The SEE should protect their input data from being sniffed or tampered with by malicious applications, such as key loggers.

- *Protected Output.* The integrity and the confidentiality of the output data are protected. Protected input and output do not only concern user interface. Indeed, they also concern exchanged data via communication interfaces, such as network interface.

- *Attestation.* The SEE should provide mechanisms allowing secure applications to authenticate themselves to remote trusted parties.

6.5.2 Security Criteria

In dual-EE, isolation, which is an essential task to implement, is provided by the SK layer. Schemes are evaluated whether the design principles of the SK layer [AFHOT06] are implemented in software or hardware in order to ensure:

- *Data Separation.* Data within one partition, namely execution environment, cannot be read or modified by other partitions.

- *Information Flow Control.* Communication between partitions cannot occur unless explicitly permitted by the SK layer.

- *Sanitization.* Shared resources cannot be used to leak information into other partitions.

- *Damage Limitation.* Security breach in one partition cannot spread to other partitions.

6.5.3 Deployability Criteria

The dual-EE approach is intended to be implemented in a real context. Thus, we evaluate how easy schemes can be deployed. Deployability criteria are numerous. In our study, we only consider the following properties:

- *Support of Legacy Systems.* We evaluate the amount of modifications needed for the GPEE to run on the underlying SK layer. Ideally, no modification, except for the inter-EE communication driver, is required.

- *Cost.* The addition of any software architecture has a cost. We only evaluate the extra silicon cost that the scheme generates. For instance, the addition of hardware module or internal processor extensions are factors which make schemes costly.

- *Overhead.* Schemes should have minimal impact on applications that do not require tamper-resistant protection. They should not incur too much overhead to the SEE either.

- *SEE Performance.* We evaluate how fast the SEE could execute complex operations.

Throughout the chapter, for brevity and consistency, each criterion is referred to with an italicized mnemonic title. In our study, we will rate each solution based on its capability to offer the criteria described above.We emphasize that it would be naive to rank dual-EE solutions simply by counting how many criteria each satisfies. Some criteria clearly deserve more weight than others. In our study, we do not suggest any weights, since providing appropriate weights depend strongly on the specific goal for which the dual-EE solutions are being compared. Indeed, our main goal here is not to provide a comprehensive exhaustive list of criteria, but instead to define a representative framework that methodologically characterizes all dual-EE solutions.

6.6 Comparative Evaluation

We now use our criteria to evaluate three different solutions of the dual-EE approach. However, we consider that studying one particular solution for each category is enough to show the interest of our comparative framework. We emphasize that, in selecting a particular solution, we do not necessarily endorse it as better than alternatives–merely that it is reasonably representative, or illuminates in some way what the category can achieve.

6.6.1 External Hardware Module: Secure Element

As said in section 2.4.2 of chapter 2, a Secure Element (SE) is essentially a minimal computing environment composed of a CPU, ROM, EEPROM, RAM, and I/O port. It is capable of running applications (called applets or cardlets) with a high level of security. In smartphones, Secure Elements come in several flavors. They could be implemented either by an embedded smart card chip, in an SD card that could be inserted in the device, or in the SIM/UICC which is used by mobile operators to authenticate subscribers to their network. In most cases, the SEE consists of Java Card OS, and the GPEE can be any commodity mobile operating system.

Functional Criteria. Secure Elements physically shield the SEE from all types of software attacks coming from the GPEE. Thus, no interference is possible during the execution of secure applications. Moreover, tamper-resistant hardware prevents protected data from being extracted by hardware attacks like microprobing and fault generation. To sum up, Secure Elements provide *protected execution* and *sealed storage*. *Attestation* is guaranteed, since only authenticated code can run in the SEE. However, Secure Elements fail to provide *protected input* and *protected output*. In practice, Secure Elements are designed in a way that there is no direct communication link with the user I/O devices. Secure Elements, for instance, cannot control user interface to allow users to securely enter their PIN code.

Security Criteria. Regarding the SK layer, it is almost implemented in hardware. Both execution environments, namely the GPEE and the SEE, run in two different CPU with their own memory and I/O devices. As a result, the software part of the SK layer does not need to take care of either *data separation* or *sanitization*. However, *damage limitation* depends on how well the inter-EE communication is controlled. The *information flow control* is implemented in the SEE. In fact, the SEE includes the SK part which is responsible for protecting the SEE from accidental or malicious communication attempts that violate the system policy.

Deployability Criteria. For reasons of silicon *cost*, Secure Elements are often made with limited resources. Indeed, an additional CPU increases the power consumption and the global cost of the device. Cost and power consumption constraints lead to design Secure Elements with limited processing power, slow processing speed and small permanent and temporary memory [WFM+07]. Therefore, secure applications have *low performance* and cannot perform complex computations. It is worth noting that Secure Elements do not actually follow Moore's law as their computation power has only changed a little since 2003 [Mit03, NXP16]. As for other criteria, Secure Elements clearly support *legacy systems* and incur *no overhead* to the SEE.

6.6.2 Bare-Metal Hypervisor: KVM/ARM

KVM/ARM is the ARM hypervisor in the mainline Linux kernel [DN14]. It is the first hypervisor to leverage ARM hardware virtualization support to run unmodified operating systems on ARM hardware. It builds on KVM and leverages existing infrastructure in the Linux kernel. KVM/ARM is a hosted bare-metal hypervisor, where the hypervisor is integrated with a host kernel. It runs the hypervisor in normal privileged CPU modes to leverage existing OS mechanisms

without modification, while at the same time leveraging hardware virtualization. In contrast to standalone bare-metal hypervisors (e.g. Xen), it supports a wide range of ARM devices despite the fact that there is no standard hardware in the ARM world. In dual-EE, the two execution environments (GPEE and SEE) are two virtual machines on the underlying hypervisor.

Functional Criteria. Hypervisors provide isolation properties to prevent potentially malicious VM (GPEE) from attacking another VM (SEE). However, hypervisors only defend against software-based attacks and do not take hardware attacks into account. This isolation property works fine for data centers, but the threat model of mobile devices includes hardware attacks. We illustrate the threat by two examples. First, an attacker with physical access to the system can read any data present in memory using the cold boot attack [HSH+09]. This attack is based on the fact that RAMs retain their contents for several seconds after power is lost. Second, an attacker with access to the system disk can run a modified version of KVM/ARM integrating malicious introspection mechanisms to snoop on the runtime states of the SEE. Thus, the KVM/ARM hypervisor provides *protected execution*, but not *sealed storage* because encryption keys can be retrieved using, for instance, the cold boot attack. Furthermore, it defines several mechanisms to provide I/O virtualization and interrupt virtualization. Therefore, it provides *protected input* and *protected output*. The KVM/ARM alone does not provide *attestation*; trust anchors, such as TPM, are required. It is worth noting that any person who has physical access to the mobile device can easily clone the SEE and capture its internal states. This might result in serious attacks, such as rolling back security updates, thus leaving the system vulnerable.

Security Criteria. Regarding the SK layer, it is *entirely implemented in software*. All of its design principles are performed by the KVM/ARM hypervisor. Therefore, the hypervisor must be tamper-resistant and evaluable. To the best of our knowledge, KVM/ARM is the smallest bare-metal hypervisor. It is comprised of only 12,883 lines of code. However, it is still too big to be formally verified.

Deployability Criteria. For KVM/ARM, platforms with hardware virtualization capabilities are required. Hardware-based virtualization is not supported on all platforms. Therefore, it presents an *additional cost* to the system. The fact that KVM/ARM leverages hardware virtualization support presents two advantages. First, it can run *legacy systems*, unlike hypervisors based on para-virtualization. Second, the incurred overhead is minimal in comparison with other virtualization solutions. For example, it achieves low overhead of I/O performance with very little implementation effort. However, KVM/ARM still generates within 10% of *overhead* over a multicore. In smartphones, where battery is still limited, 10% of overhead is not negligible.

6.6.3 Special Processor Extensions: TrustZone

ARM TrustZone technology can be seen as a special kind of virtualization with hardware support for memory, I/O and interrupt virtualization [ARM09]. This virtualization enables ARM core to provide an abstraction of two virtual cores (VCPUs): secure VCPU and non-secure VCPU. The monitor is seen as a minimal hypervisor whose main role is the control of information flow between the two virtual cores. In dual-EE, the SEE runs on the secure VCPU, while the GPEE runs on the non-secure VCPU. It is worth mentioning that ARM TrustZone was designed and optimized to implement the dual-EE approach. Indeed, it implements all the hardware extensions defined in [AFHOT06] and which the SK layer requires in order to work properly.

Functional Criteria. Similar to bare-metal hypervisor, ARM TrustZone provides *protected execution*, *protected input* and *protected output*, but it does not provide *sealed storage* or *attestation*. However, TrustZone is often completed with additional features, such as secure boot and

Comparison Category	**Comparison Criteria**	*Secure Element*	*KVM*	*TrustZone*
Security Requirements	Protected Execution	√	√	√
	Sealed Storage	√	×	√*
	Protected Input	×	√	√
	Protected Output	×	√	√
	Attestation	√	×	√*
Isolation Properties	Data Separation	HW	SW	HW
	Information Flow Control	SW	SW	HW
	Sanitization	HW	SW	HW/SW
	Damage Limitation	HW	SW	HW
Deployability Criteria	Legacy Systems	√	√	√
	Low Overhead	√	×	√
	Low Cost	×	×	√*
	High Performance	×	√	√

√: satisfies the criterion; **×:** does not satisfy the criterion;
√*: needs widely available additional hardware modules to satisfy the criterion;
HW: satisfied by hardware module; **SW:** satisfied by software implementation.

Table 6.1: Summary of Our Comparative Evaluation of Dual-EE Solutions

root of trust (RoT) hardware module, which allow TrustZone to satisfy all the requirements of a tamper-resistant environment.

Security Criteria. Regarding the SK layer, it is *mainly implemented in hardware*. The software components to be trusted are minimal, hence *evaluable*. For instance, most CPU registers are banked. Thus, saving and restoring CPU registers are performed by the processor. In addition, TrustZone enables the co-existence of cache entries of both SEE and GPEE. Thus, cleaning the cache memory during a context switch is not required. As a result, the *sanitization* process performed during a context switch is both fast and secure, since it is almost done by the hardware. Moreover, *Data flow* is well controlled. To enter the secure world, only a well-defined set of interfaces exists. Indeed, any transition between the two worlds must go through the monitor mode. This allows the SK layer to satisfy the *completeness* engineering principle.

Deployability Criteria. TrustZone incurs a limited execution *overhead*. The performance is nearly native because both execution environments can directly access their corresponding resources without going through an abstraction layer. Moreover, it can run *legacy systems* without modifications, since each world has its own user and privileged modes, thereby removing the necessity of instruction emulation. It is true that TrustZone presents an additional *cost* as it requires some modifications to the core processor, but these modifications are already extensively deployed and implemented in a wide range of ARM platforms. We note that 50% of smartphones already run on ARMv8-A which includes all the required TrustZone extensions without any cost [ARM15b].

6.7 Discussion

A summary of our comparative evaluation is presented in Table 6.1. We note that the size of the SK layer is directly proportional to the number of the isolation properties implemented in software [Gas88]. A small SK is better for security because the property of verifiability cannot be satisfied when the SK layer is too complex. Therefore, solutions with many isolation properties

provided by hardware are considered *better* than those implementing their SK layer in software.

To our surprise, bare-metal hypervisors achieve the lowest score in our framework. We did not expect this result, since the literature is abundant of solutions presenting hypervisors as a promising approach to improve system security [DN14, GPHB11, HSH+08]. In this chapter, we showed that this approach inherently suffers from three main shortcomings. First, hypervisors come from the world of data centers, and therefore their threat model does not include stolen devices. Even simple physical attacks, like cold boot attacks, can compromise the privacy requirement of tamper-resistant execution. Second, the isolation properties are entirely implemented in software, thereby negatively impacting the verifiability characteristic of the SK layer. Third, although dedicating the whole virtualization layer to hosting security tools present numerous advantages, it is not practical because it will deprive the system from using other virtualization capabilities. Furthermore, it is true that hardware-based virtualization solutions produce better overhead and fewer modifications to existing systems compared to para-virtualization solutions. However, they require specific extensions that are not supported on all platforms. For example, the widely-used Qualcomm Snapdragon MSM8974 and APQ8084 processors do not implement the hypervisor extension.

On the contrary, external hardware modules achieve the highest score in terms of security. Our results are expected, as these modules provide a confined execution environment which protects the application's authenticity, integrity and privacy against even sophisticated physical attacks. Nevertheless, external hardware modules do not fit to a certain kind of secure applications that need user interaction and better processing speed.

As for ARM TrustZone, it comes close to perfect score. Our results are consistent and expected because TrustZone implements all the hardware extensions that the SK layer requires in order to work properly. TrustZone provides a balanced trade-off between bare-metal hypervisors and external hardware modules. Indeed, it does not resist against some physical attacks and it requires a part of the SK to be implemented in software [ARM09]. In addition, TrustZone does not provide sealed storage and attestation without additional hardware modules. However, it is more secure than solutions based on bare-metal hypervisors and more flexible than those based on external hardware modules. Our framework shows that TrustZone-based solutions are efficient for real contexts. Once again, our results are consistent with existing work. At present, millions of devices integrate TrustZone-based technologies. Examples are ObC in Lumia phones [KEAR09], TIMA/TZ-RKP in Samsung smartphones [ANS+14], and <t-base of Trustonic [Tru16].

6.8 Conclusions

In this chapter, we revisited the dual-EE approach, a model that allows mobile smart devices to guarantee a tamper-resistant execution for highly sensitive applications. We introduced the dual-EE approach in the context of trusted computing. This is due to the fact that the domain of trusted computing gives us convenient abstract models to better represent the characteristics of the dual-EE approach.

In this chapter, we also provided a general classification of the dual-EE solutions defined in the literature. The goal of this classification is not to provide an extensive survey, but to examine our framework by applying it on a representative of each class. Our results are consistent with related work and sometimes unexpected. They show that TrustZone provides a balanced compromise to implement the dual-EE approach. They also show that systems requiring the maximum level of security should adopt external hardware modules, while hypervisors are ill-adapted to provide high assurance security even though they might improve the overall security level of the system.

Despite being theoretic, our results are similar to those obtained by evaluating real implementations. It is true that our comparative study has interest in its own, but we emphasize that the value of our work is to provide more insight. Indeed, our abstract model forms a theoretical

basis that systematizes the design of dual-EE solutions regarding primitives defined in the MILS architecture. We argue that a thorough understanding of the dual-EE approach allows system designers to construct better solutions without having to go through inconsistent lists of various security technologies.

We believe that our work can be easily extended to include other comparison criteria. An interesting aspect is the scheduling techniques present on MILS. In some smart embedded devices, it is necessary that malicious allocation of hardware resources (e.g. CPU time) do not impact the SEE execution. Despite their high importance, temporal constraints, unfortunately, are rarely taken into account in dual-EE solutions. We illustrate the consequence of such an absence by an example. Let us consider a smartphone integrating a dual-EE solution to execute its sensitive applications. Applications related to make phone calls are never regarded as sensitive ones because of their legacy nature, and therefore they run inside the GPEE (i.e. the main OS). Too often, these applications are executed with a high priority, since smartphones are basically still mobile phones. This means that the mobile device should interrupt any executed application to answer a phone call. However, such a high execution priority comes in contradiction with security properties offered by the SEE. Indeed, the SEE may not be able to accomplish its secure tasks if it gets frequently interrupted by the GPEE. We note that some of the secure tasks should also be executed with a high priority because of their real-time nature. Thus, if the SK layer does not implement a clear scheduling policy between the SEE and the GPEE, then a malicious piece of software could cause a Denial-of-Service (DoS) attack against the SEE by just, for instance, making phone calls. Therefore, some fine-grained secure scheduling is required for the dual-EE approach, especially when the two execution environments share the same physical resources for computation.

Chapter 7

Trusted Execution Environment: What It Is, and What It Is Not

"When I use a word, Humpty Dumpty said, in rather a scornful tone, it means just what I choose it to mean– neither more nor less."

– Lewis Carroll, Through the Looking-Glass

Contents

7.1 Introduction . . . 116
- 7.1.1 Inconsistent Definitions . . . 117
- 7.1.2 Previous Attempts . . . 118

7.2 Background . . . 118

7.3 Trusted Execution Environment . . . 119
- 7.3.1 Prerequisite: Separation Kernel . . . 119
- 7.3.2 Definition . . . 119
- 7.3.3 Discussion . . . 119
- 7.3.4 How Trust Can Be Measured . . . 120
- 7.3.5 Related Concepts . . . 121

7.4 Building Blocks . . . 121

7.5 Inter-Environment Communication . . . 124
- 7.5.1 GlobalPlatform TEE Client API . . . 124
- 7.5.2 Trusted Language Runtime (TLR) . . . 125
- 7.5.3 SafeG Real-Time Secure RPC . . . 125

7.6 Trusted I/O Path . . . 125
- 7.6.1 GlobalPlatform Trusted User Interface API . . . 125
- 7.6.2 VeriUI . . . 126
- 7.6.3 TrustUI . . . 126

7.7 Secure Storage . . . 126
- 7.7.1 GlobalPlatform Trusted Storage API . . . 126
- 7.7.2 Android KeyStore . . . 127

7.8 Formal Methods . . . 127

7.9 ARM TrustZone-Based TEE . . . 128
- 7.9.1 Industrial TEEs . . . 128
- 7.9.2 Academic TEEs . . . 129

7.9.3 TrustZone Emulation Frameworks 130
7.9.4 Comparative Study 130
7.10 TEE Applications 130
7.10.1 Ticketing 130
7.10.2 Mobile Payment 132
7.10.3 Media Content Protection 132
7.10.4 Authentication 132
7.10.5 NFC 132
7.10.6 Trusted Sensors 133
7.10.7 Trusted Platform Module 133
7.10.8 Self-Protection 133
7.11 Attacks on TEE 134
7.12 Conclusion 134

Nowadays, there is a trend to design complex, yet secure systems. In this context, the Trusted Execution Environment (TEE) was designed to enrich the previously defined trusted platforms. TEE is commonly known as an isolated processing environment in which applications can be securely executed irrespective of the rest of the system. However, TEE still lacks a precise definition as well as representative building blocks that systematize its design. Existing definitions of TEE are largely inconsistent and unspecific, which leads to confusion in the use of the term and its differentiation from related concepts, such as secure execution environment (SEE). In this chapter, we propose a precise definition of TEE and analyze its core properties. Furthermore, we discuss important concepts related to TEE, such as trust and formal verification. We give a short survey on the existing academic and industrial ARM TrustZone-based TEE, and compare them using our proposed definition. Finally, we discuss some known attacks on deployed TEE as well as its wide use to guarantee security in diverse applications. The findings of this chapter were published in [SAB15d] that was presented at the 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TRUSTCOM 15).

7.1 Introduction

Smartphones are increasingly being used not only to carry out several common tasks simplifying their users daily lives, but also to perform some sensitive applications such as those related to mobile payment. Therefore, users requirements for security are becoming more demanding. As previously explained in previous chapters (readers could refer to chapters 2 and 6), it is far from trivial to ensure a high level of security for smartphones. Indeed, new challenges arise, since modern smartphones are becoming more complex and more connected. Traditional security technologies based solely on software or implying extensive modifications to existing systems can meet neither the security nor the usability requirements of smartphones. This explains the recent trend to integrate trusted computing concepts into different systems, such as embedded systems [ABN+12].

Trusted Computing was defined to help systems to achieve secure computation, confidentiality and data protection. Primarily, trusted computing relies on a separate hardware module that offers a functional interface for platform security. The trusted platform module (TPM) [Gro14] allows a system to provide evidence of its integrity and to protect cryptographic keys inside a tamper-evident hardware module. However, the main shortcoming of the TPM is that its functionality is reduced to a predefined set of APIs, thereby not being able to offer an isolated environment to execute an arbitrary code provided by a third-party. In addition, TPM is barely suited for mobile devices, since it was originally defined to be deployed into desktop systems which greatly differ from smartphone ones (see section 2.2.2).

These limitations of the current trusted computing solutions have spurred several research work to design alternative trusted security architectures. A progressively popular approach to address trusted computing is to allow the execution of arbitrary code within a confined environment that provides tamper-resistant execution that ensures tamper-resistant execution to its applications. In the literature, many names exist to this environment. Examples include closed-box VM [GPC+03], operator virtual machine (OVM) [Fin05], TrustZone software (TZSW) [ARM09], trusted language runtime [SRSW14] and isolated computing environment (ICE) [SSW+15]. In this chapter, we are going to refer to this environment using the term coined by GlobalPlatform in [Glo11b], that is trusted execution environment (TEE).

The TEE is a secure, isolated processing environment, consisting of memory and storage capabilities [AEK+14]. It protects the integrity and the confidentiality of its running applications that, hereafter, we call *trustlets*. Numerous device manufacturers have already integrated a TEE into their smartphones [And16e, ANS+14, Tru16] and IoT devices [Seq16a]. The TEE trend is even reaching the most widely deployed smartphone system, that is Android. Indeed, the Android security chief Ludwig has announced during the Google I/O 2016 that TEE-based KeyStore (refer to section 2.3.4 for more details about the KeyStore) will be mandatory for Android Nougat [Arm16b]. Moreover, the term TEE is being heavily used in advertisements of chip vendors and platform providers (e.g. Qualcomm [Qua16a] and Solacia [Sol16]). The interest towards TEE is growing fast as several manufacturers are starting to provide some educational materiel about this technology by freely offering an open-source TEE implementation that can run on cheap chips like Raspberry PI 3 [Seq16b]. Curious readers can check whether their Android smartphones already integrate a TEE implementation by using the app "TEE Checker" [Hot16].

7.1.1 Inconsistent Definitions

The recent success history of TEE is the result of ten years of efforts on this field. As a matter of fact, the first deployed system with TEE appeared almost a decade ago, and it was demonstrated by a joint venture of Orange, Trusted Logic and STMicroelectronics [PHY05]. Despite its wide industrial deployment, little work has been done to push our knoweldge about TEE further. Indeed, no common and precise understanding for this term has been established so far, and no framework has been proposed to evaluate and compare TEE solutions. To underline this observation, we cite, in a chronological order, four definitions of TEE demonstrating the inconsistent use and understanding of the term:

1. Ben Pfaff, *Terra*, 2003 [GPC+03] The TEE is a "dedicated closed virtual machine that is isolated from the rest of the platform. Through hardware memory protection and cryptographic protection of storage, its contents are protected from observation and tampering by unauthorized parties."

2. OMTP, *Advanced Trusted Environment*, 2009 [OMT09] "The TEE resists against a set of defined threats and satisfies a number of requirements related to isolation properties, life cycle management, secure storage, cryptographic keys and protection of applications code."

3. GlobalPlatform, *TEE System Architecture*, 2011 [Glo11b] "The TEE is an execution environment that runs alongside but isolated from the device main operating system. It protects its assets against general software attacks. It can be implemented using multiple technologies, and its level of security varies accordingly."

4. Jonathan M. McCune, *Trustworthy Execution on Mobile Devices*, 2013 [VOZ+12] "The set of features intended to enable trusted execution are the following: isolated execution, secure storage, remote attestation, secure provisioning and trusted path."

All definitions somehow mention isolated execution and secure storage. In these two points there appears to be some consent. However, definitions (1) and (3) are less explicit about secure storage. Definition (1) specifically states cryptographic protection as the only means to achieve secure storage, whereas definition (3) tries to capture secure storage in a generic way as a 'protection of assets'. In addition, definition (1) describes isolation as the protection of the integrity and confidentiality of the TEE runtime states. Regarding the required security level, definitions largely differ from each others. Definitions (1) and (4) do not specify a threat model for the TEE. Definition (3) vaguely includes all software attacks in the threat model, while definition (2) clearly specifies the threats against which the TEE must resist. Definition (1) describes the TEE as a 'dedicated closed virtual machine', while the other definitions do not provide any detail about the nature of the execution environment. Some definitions are concerned by particular properties. For instance, definition (2) and McCune's definition (4) involve content management by indicating that TEE should remotely manage and update its data in a secure way (secure provisioning). Furthermore, the McCune's definition is specific to the context of human-interface devices, and therefore it includes the requirement of interaction between the TEE and end-users. This requirement was specified in the definition by using the term 'trusted path'.

We argue that existing definitions of TEE fail to capture the core aspects of this term in a clear and unambiguous manner and are even contradictory in some parts. To address this issue, in this chapter, we propose a new refined definition of TEE considering its core aspects and the 'separation kernel' trusted model (section 7.3). Thereby, we differentiate TEE from some related concepts. Moreover, we present the building blocks that capture the TEE design, followed by extensive discussion about the different properties of each building block. In this chapter, we also revisit several industrial TEE solutions in the light of our new definition in order to provide more insight.

7.1.2 Previous Attempts

This chapter might be seemed as a survey on the domain of trusted execution environment. We argue that our approach here is distinguished from existing surveys. The closest work to ours are [AEK+14] and [AGT14]. In [AEK+14], authors discuss trust computing in mobile devices. They focus on existing technologies and fail to provide a theoretical framework for TEE. As for Arfaoui et al. [AGT14], they present TEE uniquely from the GlobalPlatform perspective. In this chapter, we will show that this model is limited compared to other TEE existing advanced models.

The particularity of our work is that we present a refined definition of TEE. Its core properties are clearly defined. Other important related topics are discussed, such as formal verification, known attacks, and a classification of the proposed applications in the literature using TEE to guarantee security.

7.2 Background

In order to better understand this chapter, readers are required to be familiar with some background information:

- Smartphone security and its security challenges (section 2.2);
- ARM TrustZone (section 2.4.3);
- The dual-execution-environment approach (chapter 6).

7.3 Trusted Execution Environment

In this section, we first briefly recall the model of 'separation kernel', which is a fundamental concept related to the dual-execution-environment approach, and thus to the TEE. We then present a new refined and comprehensive definition, as well as analyze its core aspects. Finally, we distinguish the TEE from some related terms: secure execution environment (SEE) and dynamic root of trust measurement (DRTM).

7.3.1 Prerequisite: Separation Kernel

The separation kernel is a foundation component of the TEE. It is the element that assures the property of isolated execution. The separation kernel, firstly introduced in [Rus81], is a security kernel originally used to simulate a distributed system. Its main design purpose is to enable the coexistence of different systems requiring different levels of security on the same platform. Basically, it divides the system into several partitions, and guarantees a strong isolation between them, except for the carefully controlled interface for inter-partition communication.

The security requirements for separation kernels are described in the *Separation Kernel Protection Profile* (SKPP) [Inf07]. The SKPP defines separation kernel as "hardware and/or firmware and/or software mechanisms whose primary function is to establish, isolate and control information flow between [...] those partitions". Unlike traditional security kernels, such as operating systems, micro-kernels and hypervisors, the separation kernel is quite simple providing both time and space partitioning.

The security requirements are composed of four main security policies (similar to the security criteria presented in section 6.5.2):

1. *Data (spatial) separation.* Data within one partition cannot be read or modified by other partitions;
2. *Sanitization (temporal separation).* Shared resources cannot be used to leak information into other partitions;
3. *Control of information flow.* Communication between partitions cannot occur unless explicitly permitted;
4. *Fault isolation.* Security breach in one partition is contained into its partition, thereby cannot spread to other partitions.

7.3.2 Definition

Trusted Execution Environment (TEE) is a tamper-resistant processing environment that runs on a separation kernel. It guarantees the authenticity of the executed code, the integrity of the runtime states (e.g. CPU registers, memory and sensitive I/O), and the confidentiality of its code, data and runtime states stored on a persistent memory. In addition, it shall be able to provide remote attestation that proves its trustworthiness for third-parties. The content of TEE is not static; it can be securely updated. As for its threat model, the TEE resists against all software attacks as well as the physical attacks performed on the main memory of the system. Attacks performed by exploiting backdoor security flaws are not possible.

7.3.3 Discussion

We define TEE as an execution environment that protects both its runtime states and stored assets. This requirement implies the need for isolation and secure storage. Isolation is enforced by

the underlying separation kernel, while secure storage is guaranteed by some kind of *trust anchor*. Unlike dedicated hardware coprocessors (refer to section 6.4.2), TEE is able to flexibly manage its content by installing or updating its code and data. Moreover, it must define mechanisms to securely attest its trustworthiness to third-parties, hence the richness of the TEE execution model. The threat model is also specified. It includes a powerful adversary that is able to compromise the system main OS and execute an arbitrary code to breach the TEE security. It also includes the physical attacks performed on the main memory and its non-volatile memory (i.e. secure storage). This is ensured by prohibiting a direct access to the runtime memory in order to provide a high level of security without degrading the execution performance. Regarding security backdoor flaws, we admit that the absence of such kind of flaws constitute a strong assumption. However, being a *trusted* environment, we require that the TEE code be analyzed by some trusted parties that scrupulously search for backdoors.

We argue that our definition is more general and includes all the previously presented definitions of TEE. Conceptually, our definition means that no untrusted code can cause, enable, or prevent any event in the TEE. Events are not only defined by the execution of instructions, but also by traps, software exceptions and hardware interruptions. We construct our definition so that *Secure execution*, *openness* and *trust* constitute its main parts. 'Openness' is characterized by dynamically updating the core TEE system. For 'Secure execution' and 'Trust', they are discussed further in the next subsection.

7.3.4 How Trust Can Be Measured

As mentioned in the above definition, and as its name indicates, the concept of **trust** is crucial to the TEE. Thus, a direct comparison between two systems in terms of TEE is only possible if trust can be quantified. The main problem is that trust is a subjective property, hence non-measurable. In English, trust is the "belief in honesty and goodness of a person or thing". A belief is hard to capture in a quantified way. The notion of trust is more subtle in the field of computer systems. In the real world, an entity is trusted if it has behaved and/will behave as expected. In the computing world, trust follows the same assumption.

In computing, trust is either static or dynamic. A **static trust** is a trust based on a comprehensive evaluation against a specific set of security requirements. For instance, the Common Criteria (CC) [Com16b] are an international standard that provides assurance measures for the security evaluation. The CC specify seven evaluation assurance levels (EAL1–-EAL7), where levels with higher numbers include all requirements of the preceding levels. In static trust, the trustworthiness of a system is measured only once and before its deployment. **Dynamic trust** is quite different. It is based on the state of the running system, and thus it varies accordingly. A system continuously changes its "trust status". In dynamic trust, the trustworthiness of a system is constantly measured throughout its life cycle.

The concept of dynamic trust is based on the existence of a secure and reliable means that provides evidence of the trust status of a given system. Trust, in this context, can be defined as an expectation that the system state is as it is considered to be: secure. This definition requires a trusted entity called **Root of Trust (RoT)** to provide trustworthy evidence regarding the state of a system. The role of RoT is divided into two parts. First is the **trusted measurement** and second is the function that computes the **trust score**. The trustworthiness of the system, namely the generated score, depends on the reliability of the trust measurement. If a malicious entity can influence the trust measurement, then the generated score of trustworthiness is of no value. Therefore, RoT is necessarily a tamper-resistant hardware module. RoT, sometimes called trust anchor, can be implemented using various technologies. This depends on the hardware platform that is used to guarantee the isolation properties in the separation kernel. For instance, TrustZone-based systems [WFM+07] rely on secureROM or eFuse technology as trust anchor. PUF, Physically Unclonable Function, is a promising RoT technology for TEE [ZZH+14].

Trust in TEE is a hybrid trust; it is both static and semi-dynamic. Before deployment, a TEE must be certified by thoroughly verifying its security level in accordance of a *protection profile*, a document that contains a predefined set of security requirements. For instance, GlobalPlatform defines a protection profile that conforms to EAL2 [Glo15b]. In addition, during each boot, the RoT assures that the loaded TEE is the one certified by the platform provider. Strictly speaking, RoT protects the integrity of the TEE code. Once running, the integrity is protected by the underlying separation kernel. The trust in TEE is considered semi-dynamic because the TEE is not supposed to change its trust level while running because it is protected by the separation kernel. In this model of trust, the trust measurements are integrity measurements, and the trust score is a boolean that indicates the integrity state of the code. The TEE is trusted when its trust score is true, untrusted otherwise. The quality of the trust score depends on the defined measurements for integrity. To evaluate the actual trust, as a first step, we define a **trust function** f(`TEE`; `protection profile`; `RoT`; `measurements`) as a function that returns the trust level of a given TEE depending on three parameters: the certificating protection profile, the reliability of RoT, and the integrity measurements. The precise definition of this function is beyond the scope of this thesis.

7.3.5 Related Concepts

In this section, we highlight the conceptual differences between TEE and the related terms SEE and DRTM.

Secure Execution Environment (SEE) is a prerequisite for TEE, but it does not consider trust aspects. SEE is a processing environment that guarantees the following properties [SLS+05]: (1) *authenticity*: the code under execution should not have been changed; (2) *integrity*: runtime states should not have been tampered with; and (3) *confidentiality*: code, data and runtime states should not have been observable by unauthorized applications, or even by the main OS of the system. In contrast to TEE, the design of SEE does not involve RoT to assert the integrity and authenticity of the loaded code. Moreover, the SEE definition of [SLS+05] does not specify secure mechanisms to update its applications and confidential data. Stated all the above observations otherwise, our definition of section 7.3.2 implies that TEE is an *open* SEE that guarantees *trust*.

Dynamic Root of Trust Measurement (DRTM) is a group of techniques that enables some pieces of code to be executed in an isolated environment, without trusting the previously loaded software. This technology has been used to securely execute critical software applications. Unlike TEE, the trusted computing base (TCB) of DRTM is not limited to the code running in the isolated environment, but it also includes a small part of the main OS. Moreover, DRTM does not provide secure storage or include trusted mechanisms, such as remote attestation and integrity measurement, in its core design. In contrast to the TEE definition stated above, the isolated execution environment needs to use the main memory as its runtime memory, and hence is vulnerable to attacks on main memory. TEE is supposed to withstand such kind of attacks by executing its code in a protected memory. Another noticeable difference is that DRTM does not allow concurrency. All software is frozen until the end of the isolated environment execution [Wil13]. Therefore, DRTM is not suitable for systems which execute non-secure applications with real-time constraints.

7.4 Building Blocks

To capture the core design aspects of TEE, we propose the following definitions which are illustrated in Figure 7.1.

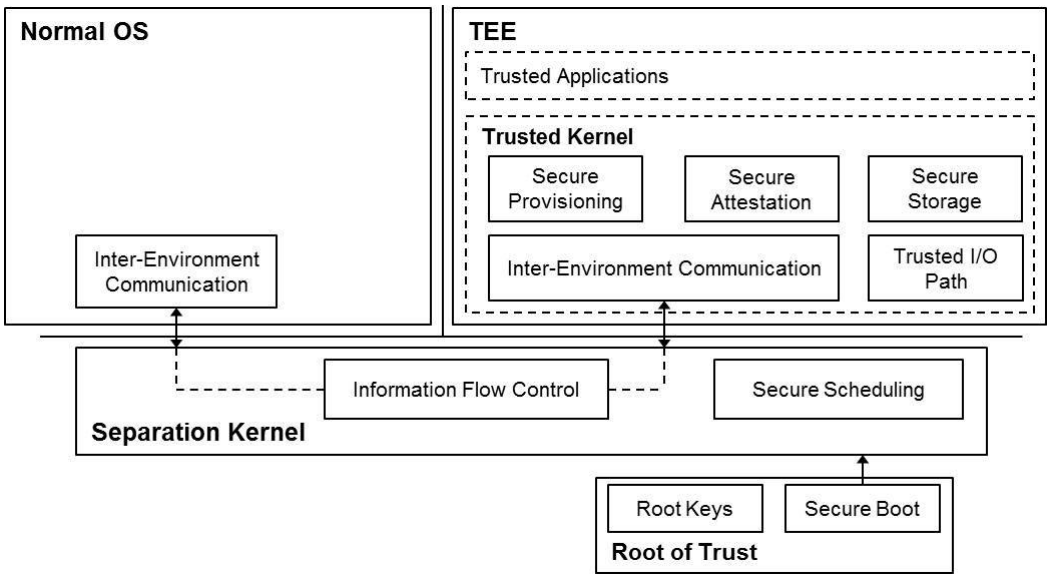


Figure 7.1: An Overview of TEE Building Blocks

• Secure Boot assures that only code of a certain property can be loaded. If a modification is detected, the bootstrap process is interrupted. An example implementation of secure boot, as proposed by Arbaugh et al., is to verify the integrity of a succeeding component according to a given reference value [AFS97]. Generally speaking, the design of secure boot consists of various stages, and therefore, a chain of trust is established. This chain can be represented by the recurrence:

$$I_0 = True;$$
$$I_{i+1} = I_i \wedge V_i(L_{i+1})$$

where I_i denotes the integrity of layer i and V_i is the corresponding verification function. There are several possible definitions of V_i. For instance, it performs some cryptographic hash of the $(i+1)$th layer, and compares the result to the reference value. We note that without the integrity of the initial boot code, represented by the I_0, any further integrity verification becomes pointless. This implies the high importance of the initial boot, and this is the reason why it is often protected by a tamper-evident hardware module.

• Secure Scheduling assures a "balanced" and "efficient" coordination between the TEE and the rest of the system. Indeed, it should assure that the tasks running in the TEE do not affect the responsiveness of the main OS. Thus, the scheduler is often designed preemptive. Furthermore, the scheduler should take real-time constraints into consideration. Authors in [SHT12] propose a secure scheduler that enhances the responsiveness of the main OS without compromising the real-time aspect of the system.

• Root of Trust (Rot), often called trust anchor, provides a way to attest the authenticity and the integrity of the code running inside the TEE. Generally, this attestation happens by verifying a signature that was previously computed over the running software. Either the signature or the private key used to generate the signature is securely stored inside the RoT. At present, trusted systems rely on secureROM (a read-only memory) or eFuse (a one-time programmable memory) technologies as trust anchor. Once the signature key is written, no modification is possible, even though key update is desirable in practice.

In this context, PUF, Physically Unclonable Function, is a promising technology. PUF has the ability to generate a unique-per-device key. This key is used for authentication and secure storage of cryptographic secrets. PUF has several advantages. It provides more flexibility for the TEE, since it allows the related system to change the signature key even after manufacturing. In addition, the key generated by the PUF is unique. Consequently, class attacks are not possible. This means that an attack on a device cannot trivially impact other devices. Moreover, no secure non-volatile memory is required to store secrets.

To the best of our knowledge, authors of [AP12] are the first to present the idea of integrating a PUF inside smartphones. In their design, PUF generates AES keys that are used to perform secure boot as well as disk encryption. However, authors only discuss their theoretical design and how it can be implemented by FPGA technology, without providing any implementation that shows the feasibility of their solution. As for Zhao et al. in [ZZH+14], they implement their proposed PUF-based solution and present a thorough evaluation related to their design.

• Inter-Environment Communication defines an interface allowing TEE to communicate with the rest of the system. Due to the strong isolation between the TEE and the rest of the system, this inter-environment interaction is considered as a Remote Procedure Call (RPC). It is usually implemented over a traditional RPC channel such as pipes, sockets or shared memories. There exist various methods and implementations of inter-environment communication. However, each mechanism should satisfy three key attributes: reliability (memory/time isolation), minimum overhead (unnecessary data copies and context switches), and protection of communication structures. These attributes define several communication models. We later discuss these models in section 7.5.

Despite its numerous benefits, inter-environment communication introduces new threats: (1) message overload attacks [RD05]; (2) user and control data corruption attacks [CXS+05]; (3) memory faults caused by shared pages being removed; and (4) unbound waits caused by the noncooperation of the untrusted part of system.

• Secure Storage is storage where confidentiality, integrity and freshness (i.e., to protect against replay attacks and to enforce state continuity [PLD+11]) of stored data are guaranteed, and where only authorized entities can access the data [LSW10]. A common way to implement secure storage is sealed storage. Sealed storage is based on three components: (1) integrity-protected secret key that can be accessed only by the TEE; (2) cryptographic mechanisms, such as authenticated encryption algorithms; and (3) data rollback protection mechanism, such as replay-protected memory blocks (RPMB) (see [RPV15]).

In section 7.7, we discuss different secure storage systems that exist in the literature.

• Trusted I/O Path protects authenticity, and optionally confidentiality, of communication between TEE and peripherals (e.g., keyboard or sensors). As defined in [Dep85], the trusted I/O path is a set of mechanisms by which users can communicate directly with the secure part of the system. These mechanisms authenticate trusted software users, and vice versa. Trusted path to user-interface devices enables broader functionality within TEE. It allows a human user to directly interact with applications running inside TEE. Thus, input and output data are protected from being sniffed or tampered with by malicious applications.

To be more precise, trusted I/O path protects against four classes of attacks: screen-capture attack, key logging attack, overlaying attack, and phishing attack. The *key logging* and *screen-capture* attacks compromise the confidentiality of the input/output data. The *overlaying attack* allows malicious software to modify all or part of the screen that users see, thereby compromising the integrity of the output data. A hard problem and a very difficult attack to defend against is the *phishing attack*. It is a combination of two attacks: social engineering and frame-buffer overlay. The most common form of the phishing attack is the spoofing attack, in which malicious

software mimics the User Interface (UI) of another application. If the user is not careful enough to identify the fake UI, they may enter some secret information, like their PIN, into the malicious software.

Now, we illustrate these threats by an example. Protecting the integrity of the output data ensures that *seeing is believing*. Hence, users can trust any message displayed on the smartphone screen. For instance, they can trust the transaction information shown by their secure payment applications. Thus, users are protected from being lulled with the amount of money authorized for a transaction by showing 20$ for a transaction of 100$.

We discuss some solutions providing trusted user-interface in section 7.6.

7.5 Inter-Environment Communication

In the literature, we identify three models of communication: (1) GlobalPlatform TEE Client API [Glo10]; (2) secure RPC (Remote Procedure Call) of Trusted Language Runtime [SRSW14]; and (3) real-time RPC of SafeG [SHT13]. Secure inter-environment communication is proposed in [JKK+15].

7.5.1 GlobalPlatform TEE Client API

Client API specification within the GlobalPlatform architecture specifies a communication model with the TEE. In this model, the communication is not symmetric. It is based on the master/slave model. This implies that the direction of communication is always from the normal world to the TEE. Only the applications running inside the normal OS make use of the TEE Client API. Therefore, the library implementing the TEE Client API is executed in the normal OS.

TEE Client API defines a handful of functions for communication across the security domains. These operations can be divided into the following groups (in the order of use):

Initialize. open a context with one of the running TEE and initialize a communication session with the desired trusted application.

Prepare shared memories. allocate or register shared memories into which parameters and results will be written.

Invoke commands. perform the remote calls with the given parameters.

Finalize & Release. fetch the result of the trusted application and then release memories before closing the opened session.

The key design principles of TEE Client API are:

- API in C language: portability is the main argument of choosing the C language, since it is the common denominator for almost all smartphone systems.
- Blocking functions: the calling model of RPC is synchronous in which the caller blocks waiting for the underlying task to complete before continuing its execution.
- Support memory sharing by pointers: input and output data should be provided as memory buffers using simple C pointers. However, short messages can be passed by copy in order to avoid the overhead of sharing memory.

It is worth noting that the defined communication model is rather close to that of Secure Elements (refer to section 2.4.2) as many GlobalPlatform members that define the TEE Client API are also issuers of Secure Elements [Glo16c].

7.5.2 Trusted Language Runtime (TLR)

The inter-environment communication model of TLR is designed to make programming secure applications relatively easy. Similarly to TEE Client API, it is based on the master/slave model. However, in contrast to the GlobalPlatform model, TLR automatically manages its communication messages. Indeed, when the application in the normal OS sends a command to the TEE, the TLR library creates a transparent proxy and returns it to the caller. The proxy encodes the parameters into messages, and then it forwards them together with the method invocation request to the TEE. The request is decoded and the corresponding method is invoked on the corresponding secure application.

7.5.3 SafeG Real-Time Secure RPC

SafeG is designed to work in a real-time environment. Thus, the inter-environment communication is built to satisfy the strict reliability requirements of such an environment: throughput, memory size, and time isolation. Indeed, SafeG increases the communication throughput by minimizing the overhead caused by unnecessary data copies and context switches. To this end, all data communications occur through shared memory. In addition, it guarantees memory protection and timeliness of the real-time environment.

SafeG splits requests in two parts: the data part and the event part. This separation allows tasks to communicate using both polling and event-driven communication models. The data part involves non-blocking operations to write or read blocks of data. This is implemented using lock-free bidirectional FIFO in the shared memory. The event part involves asynchronous notifications which are implemented through inter-environment interrupts. SafeG provides a set of API which is similar to TEE Client API. However, it is not based on the master/slave model of communication.

7.6 Trusted I/O Path

In the literature, we identify four systems that leverage TEE to implement trusted interface: GlobalPlatform Trusted User Interface API [Glo13], VeriUI [LC14] TrustUI [LMH+14], and Genode TEE [GEN14].

7.6.1 GlobalPlatform Trusted User Interface API

GlobalPlatform defines a set of specifications on how trusted interface should be designed and implemented. These specifications standardize how trusted applications, as they are defined in TEE Internal API [Glo11a], protect their interactions with users. We note that touchscreen is the only concerned peripheral. The management of other peripherals, such as the camera, is not included.

The Trusted User Interface API defines a secure indicator in its specifications. GlobalPlatform does not mandate any particular technology as long as this technology satisfies the specified requirements. The only requirement for a secure indicator is to indicate the running OS, and thereby protecting against phishing attacks.

The main drawback of the proposed model is its limited defined interface. Indeed, the user interface is designed to display messages with limited information. It can be composed of only some input fields and a virtual keyboard. This allows trusted applications to display some messages to their users and also ask them to authenticate by entering their passwords. GlobalPlatform does not include animations in its model.

7.6.2 VeriUI

VeriUI is a framework providing trusted user interface for third-party cloud applications. In order to offer a generic trusted interface, a secured webkit (an engine that renders web pages) is used to display the secure interface. The use HTML pages makes the design flexible and easily adaptable to any cloud service. VeriUI works as follows: the application running in the normal OS calls the secure webkit with the cloud URL as a parameter. The webkit communicates with the cloud services that ask to attest the authenticity of the transaction request. The attestation is required, so that the remote server authenticates the secure interface. The cloud then sends HTML pages to the secure webkit that displays them to the user. In order to run VeriUI, the TEE should support various libraries: Qt as GUI lib, OpenSSL for cryptographic operations, and input engine for text entry. Moreover, the RoT is assumed to be pre-installed with a key pair by the manufacturer. This key pair is used for signature.

We note that VeriUI is vulnerable to spoofing attack in which an application requests a phishing webkit to open a phishing URL. If tricked, the user enters her password in the phishing interface, allowing the attacker to steal it. Authors propose the secure indicator, TrustZone LED for instance, as a solution to defend against such attack. Nevertheless, Rachna et al. argue that secure indicators are ineffective [DTH06].

7.6.3 TrustUI

TrustUI offers trusted user interface by combining some kind of randomization with LED lights. In TrustUI design, a device driver is split into two parts: a back-end running in the normal OS, and a front-end running in the TEE. The two parts communicate through related proxy modules that exchange their data through shared memory. This design has the advantage of supporting legacy systems by supporting their device drivers, but it comes at risk as all input/output data pass through the normal OS.

TrustUI proposes a solution to defend against the phishing attack. It leverages LED indicator and color randomization. TrustUI configures LED lights as secure peripheral and regularly changes the background color of the screen. By comparing the randomized display color with the LED color, users can attest whether the interface is indeed generated by the TEE.

7.7 Secure Storage

In the literature, we identify two trusted storage API: GlobalPlatform API for Trusted Storage [Glo11a] and key storage in Android KeyStore (section 2.3.4).

7.7.1 GlobalPlatform Trusted Storage API

GlobalPlatform defines trusted storage API for keys and general-purpose data. It also defines a set of requirements for the library implementing this API. Essentially, there are two main requirements. Firstly, the stored data must be protected by some cryptographic mechanisms, notably by authenticated encryption, since the actual storage device might be accessible by the normal OS. Secondly, the stored data must be bound to the device so that they cannot be copied to another device.

The model of GlobalPlatform trusted storage is not based on file systems. Instead, the storage unit is an object. An object is a stream of bytes that are securely written on a non-volatile memory. Each secure object has a type that defines its content. For instance, there are different object types for keys and data. Managing secure objects depends on whether they are persistent or transient. In contrast to persistent objects, transient objects are automatically deleted from the memory as soon as the session to the trusted application is closed.

Mobicore [Gie08], the TEE developed by Giesecke & Devrient GmbH (G&D) and the ancestor of <t-base of Trustonic [Tru16], implements a model of trusted storage that is close to the model of GlobalPlatform. Mobicore stores secure objects as containers [Gie15]. There are six types of containers: SoC, root, service provider, service provider data, trustlet, and trustlet data. Containers are organized into a tree-like structure with the SoC container as the root of the tree. Each container has a parent-container and a list of children. Containers, except the SoC container, are encrypted with the secret key of their parent. Secure applications are stored as trustlet containers. Trustlets store their data as trustlet data containers. Therefore, secure objects are protected by the secret key of their secure application.

7.7.2 Android KeyStore

As previously mentioned in 2.3.4, Android KeyStore ensures secure storage for cryptographic keys. The KeyStore exposes its functionalities via a standard Java interface. The interface includes several classes, notably KeyStore and KeyGenerator. As their names indicate, the KeyStore Java class is used to store keys, and the KeyGenerator class is used to generate keys. Starting from API level 18, Android defines a new security provider called Android Key Store. This JCE security provider is implemented using some hardware abstraction layers that usually rely on TEE to perform their sensitive tasks. The most popular TEE underlying the Android KeyStore is Qualcomm TEE called QSEE [Qua16a].

At the writing of this disseration, there are five classes of commands:

1. **Generate Key** to generate symmetric keys, such as AES and HMAC keys, as well as key pairs, such as RSA and EC (elliptic curve) keys;

2. **Import Key** to store new keys inside the KeyStore;

3. **Export Key** to have access to the key bytes outside the KeyStore;

4. **Sign Data** to use the stored key for signing data;

5. **Verify Data** to use the stored data for verifying signatures.

In almost all TEE-based implementations of KeyStore, the only thing that is indeed protected by TEE is some form of 'master' key encryption key (KEK). Keys that are generated by users are protected by being encrypted with the KEK. This allows for practically unlimited number of protected keys. However, if the TEE KEK is compromised, all the user-generated keys are compromised. However, we note that the KEK is unique for each device, and therefore any security breach would solely concern one device.

7.8 Formal Methods

The design of TEE, or any piece of software, consists of two aspects: requirements specification and implementation. A TEE is said to be *correct* if its implementation is verified to satisfy all the defined requirements. Formal methods, which are mathematically based languages and techniques, are used to prove correctness. Although formal methods do not *necessarily* guarantee correctness, they provide insights which prove useful in constructing better systems.

There are two goals for formal methods: specification and verification. **Formal specifications** aim at describing the requirements of a system in a syntax-based language. They are a necessary condition to perform proof-based verification on implementation. A set of formal specifications, combined with a formal language produce a *formal model*. In literature, there are several formal models for separation kernel. The most widely used formal specifications for

separation kernel are those proposed by Greve, Wilding and Vanfleet (GWV) [GWV03]. Concerning formal languages, the mainly used ones are Z notation, B method [KKMN10], HOL4 (a variation of High Order Logic) [HOL16], and ACL2 (A Computational Logic for Applicative Common Lisp) [KM16]. **Formal verification** is used to analyze the formal model for the desired properties. Two general approaches to formal verification exist in practice today. The first, *model checking*, is a technique in which systems are modeled as finite state systems. The second, *theorem proving*, proves that a system satisfies the specifications by deductive reasoning. Although proofs can be constructed by hand, machine-assisted theorem provers are used in most cases. Theorem proving is used more often than model checking because it can efficiently deal with complex properties.

We illustrate with two formally verified separation kernels.

1. INTEGRITY-178B [Ric10] is a separation kernel of Green Hills Software that is certified EAL6+. It uses GWV as formal specifications, ACL2 as formal language, theorem proving as formal verification method, and ACL2 theorem prover.

2. SeL4 [MMB+13] is developed by NICTA and was formally verified for security critical domain. It uses information flow security as formal specifications, HOL as formal language, theorem proving as formal verification method, and Isabelle/HOL theorem prover.

Formal methods play an important role in computing the 'trust level' defined by the trust function (section 7.3.4), since the protection profile could be defined using formal specifications and proved using formal verification. This could highly improve the trust level. Nevertheless, it is important to keep in mind that formal methods are not a silver bullet. The trust function has other parameters and they could negatively impact the global trust level, even though formal methods are employed. For the best of our knowledge, there is no TEE that is formally verified. We believe that formal characterization of TEE specifications will be regarded as a considerable contribution. The most difficult part will be to include all the components and building blocks in a single model, despite their heterogeneity. Any formal model must at least comprise the underlying separation kernel, the root of trust and the secure execution environment.

7.9 ARM TrustZone-Based TEE

ARM TrustZone technology can be seen as a special kind of virtualization with hardware support for memory, I/O and interrupt virtualization (section 2.4.3). This virtualization enables ARM core to provide an abstraction of two virtual cores (VCPUs): secure VCPU and non-secure VCPU. The monitor is seen as a minimal hypervisor whose main role is the control of information flow between the two virtual cores.

A short survey on the existing TrustZone-based TEE solutions in both the academic and industrial worlds is presented.

7.9.1 Industrial TEEs

Established companies have invested to define their own TEE and integrate them in their devices. Some companies have published their architecture, while some have preferred secrecy over openness. Companies which open their TEE include Nokia and Samsung. Nokia, currently Microsoft, integrates their TEE called ObC [KEAR09] into Nokia Lumia devices. Samsung define TZ-RKP [ANS+14] that is deployed on the latest Samsung Galaxy series. Closed-architecture TEEs include <t-base of Trustonic [Tru16], SecuriTEE of Solacia [Sol16], and QSEE of Qualcomm [Qua16a]. Sierraware [Sie16] proposes two versions of TEE: open-source TEE that is called SierraTEE and a licensed TEE. Trusted Foundation and MobiCore, defined by Trust Logic and

G&D respectively, are disappearing from the market, since the two companies have joined their efforts and formed Trustonic.

We also consider as industrial TEEs those which are defined by companies, but there is no public information about their actual deployment on commercial devices. STMicroelectronics, in collaboration with Linaro make their TEE called Open Portable TEE (OP-TEE) and that is available on GitHub [Bra16]. OP-TEE is compliant with GlobalPlatform specifications and it was implemented for educational purposes. Indeed, Sequitur Labs have recently announced that they ported OP-TEE to inexpensive Raspberry Pi 3 platform [Seq16b]. With this port, more developers can now begin testing their secure applications on real TEE platforms. Nvidia also proposes an open-source implementation of TEE and it calls it TLK [Nvi14]. Nvidea chooses not follow the specifications of GlobalPlatform and thus provides their own proprietary solutions. More recently, Google published some TEE specifications called Trusty TEE [And16e]. Trusty TEE mainly consists of hardware abstraction layers, and a set of libraries and APIs to handle communications between the TEE and Android. To the best of our knowledge, no Trusty-compliant TEE has been commercialized yet.

7.9.2 Academic TEEs

In the academic world, numerous TEE architectures have been proposed. Each one of them focuses on one design aspect and implements some solutions to specific problems.

1. *ViMoExpress* (defined by the Korean Electronics and Telecommunication Research Institute) [OKK+12]: it defines the mechanisms that can be used by the Separation Kernel to share hardware resources, especially peripherals, between the normal OS and the TEE;

2. *SafeG* (defined by the Japanese Nagoya University) [SHT10]: it defines a TEE that satisfies the stringent constraints of a real-time environment, including an efficient model for the inter-environment communication, and a fine-grained preemptive secure scheduling;

3. *Genode TEE* (defined by Genode Labs) [GEN14]: it defines a rich execution model in which the TEE does not follow the master/slave model , and thus it can be executed in its own. In addition, Genode TEE heavily optimizes the trusted I/O path so that the TEE can efficiently run 3D animations.

4. *Open TEE* (defined by Intel Research Institute for Secure Computing) [MDNA15]: it focuses on providing developer-friendly tools for prototyping secure applications. It is hardware independent and compliant with GlobalPlatform specifications;

5. *Andix OS* (defined at TU Graz University of Technology) [FAWH15]: it defines an easily extensible TEE in order to flexibly investigate the different protection properties offered by a TEE, such as replay-resistant secure storage [HWF15];

6. *TLR* (defined by Microsoft Research) [SRSW14]: it simplifies the developers experience by designing a customized compiler that automatically generates the proxy objects handling all the low-level communication messages between the TEE and the normal OS;

7. *TrustICE* (defined by the Chinese State Key Laboratory of Information Security) [SSW+15]: it focuses on minimizing the trusted code base of the TEE by keeping its size unchanged regardless of the number of secure applications being executed. Moreover, it simplifies the installing process of secure applications by running them inside the normal OS. Surely, their security is guaranteed.

7.9.3 TrustZone Emulation Frameworks

The introduction of TrustZone has inspired a number of security research activities in both industry and academia. Moreover, TrustZone is available in many mobile devices and embedded systems. In some cases, it is quite difficult and expensive to acquire open-source hardware platforms supporting TrustZone. In order to implement their TEE prototypes, developers often require to access and control all aspects of the platform. The lack of suitable development frameworks for TrustZone has hindered researchers in their efforts to implement and evaluate their systems. Non-disclosure agreements are another hurdle impeding the publication of scientific papers including complete and fully explained architectures. This problem is best addressed by providing an openly available emulator for TrustZone. Based on QEMU, Winter provides an open-source platform emulator which is capable of simulating system-level features of TrustZone [WWPT12]. The emulator can be downloaded at [Win13]. The emulator is complemented by a small experimental secure kernel running in the emulated secure zone and providing a basic C runtime environment. The whole emulation platform is demonstrated by running a software-based TPM developed by IBM in the secure kernel.

7.9.4 Comparative Study

We compare six TEE solutions using our proposed building blocks. An overview of these TEEs are presented in Table 7.1. We decided to compare only these TEEs because they represent the wide spectrum of the different solutions. We do not include secure boot or RoT in our comparison criteria, since Non-disclosure agreements (NDA) prevent authors from providing details about their implementations.

As illustrated in Table 7.1, only the two TEE widely deployed, namely ObC and <t-base, provide secure provisioning. Trustonic attempts to control the content management of its TEE, while ObC opens their TEE for any party to install secure applications. Similarly, due to its high complexity, secure UI is not defined for all TEE. Concerning secure storage, it is often accomplished using sealing storage with some scheme of authenticated encryption. Our comparative table shows that there is no clear standard for inter-environment communication. The use of proprietary interface is common.

7.10 TEE Applications

Now, we provide a classification of the proposed applications in the literature that rely on TEE to guarantee security. The use of TEE paves the way for offering services requiring a high level of security in a complex and connected system. TEE is used to provide a wide range of secure services: ticketing, mobile payment, media content protection, two-factor authentication and trusted platform module.

7.10.1 Ticketing

The TEE is used for ticketing systems. A ticket is a proof of purchase relating to a particular object or service. Users receive a proof of purchase from a service provider, store it on their device, and then use it to have access to the corresponding service. The TEE is used to securely store the ticket and prevent attackers from successfully forging valid tickets [WHCE05, WHEC06]. Tamrakar et al. [TEA11] define a privacy-friendly system of public transport ticketing with NFC-enabled smartphones.

TEE	Provisioning	Secure Storage	Secure UI	Inter-Environment Communication
ObC	Open provisioning, which means that the content management does not need the approval of any trusted-party.	Sealing storage using AES-EAX encryption. The root key is derived from a one-time programmable (e-Fuse) persistent on-chip key.	Defined	Proprietary interface
<t-base	Owner-centric provisioning model in which an application to be installed on TEE needs to be encrypted using a key derived from the platform secret key.	Sealing storage that is not based on file systems. Instead, the unit of storage is an object. Objects are organized into a tree-like structure. Containers are protected by the secret key of their parent.	Defined	GlobalPlatform TEE Client API
Andix OS	Not defined	Sealing storage	Not defined	GlobalPlatform TEE Client API
TLK	Not defined	Sealing storage	Not defined	Proprietary interface
TLR	Not defined	Sealing storage with mechanisms that protect against rollback attack.	Not defined	.NET Remoting
SafeG	Not defined	Unknown	Defined	Secure RPC

Table 7.1: A Comparison Study of the Six Representative TEE Solutions

7.10.2 Mobile Payment

It becomes a global trend to use smartphones as a platform for secure transactions. Pirker et al. in [PS12] define a privacy-friendly online prepaid payment framework. They define a payment trustlet running inside the TEE, and any application can communicate with its public API in order to perform payment transactions. In [PSW12], Pirker et al. define another privacy-friendly payment system that allows users to purchase and anonymously consume cloud processing resources. Li et al. in [LHSB13] define a system of transaction confirmation for online transactions by leveraging a modified version of CP-ABE schemes. Their proposed model is a special form of two-factor authentication. In [YYZ+16], authors design AEP-M: a payment scheme using divisible e-cash. Thus, AEP-M can protect users privacy while performing their transactions. In addition, the protection mechanisms of TEE enable several optimizations for the proposed e-cash scheme. Using the computation power of TEE, AEP-M is efficient as it allows users to pay in at most 450 milliseconds.

7.10.3 Media Content Protection

The design of secure multimedia applications is not trivial. It should meet the interests of both content providers in protecting their content from piracy and users in improving the overall quality of their experience. Furthermore, media processing is generally a resource intensive task. The authors of [TWP13] suggest to use the TEE as an enabler for a Secure Media Path (SMP) on mobile devices. Thus, the sensitive operations are moved into the trusted environment. The operations include content decryption, policy engines and key management. In [AFA+13], a complete multimedia content playback solution is defined and implemented. The security is enhanced by combining the (U)SIM with the TEE. The architecture comprises of content purchase, protected storage and secure content viewing.

7.10.4 Authentication

Users need to regularly authenticate themselves to remote servers in order to have access to certain services. A simple authentication based on user ID and password is easy to compromise. A popular approach is to use two-factor authentication (2FA). In [vRDP13], authors study how TEE-enabled mobile devices can be used to replace hardware tokens for 2FA. They conclude that the most problematic issue is the trusted path to the interface. Marforio et al. in [MKS+14] propose an architecture of 2FA in the context of point of sale transactions. They use the location of users devices as the second authentication factor. TrustOTP [SSWJ15] is a TEE-based one-time password solution. The TEE protects not only the confidentiality of the generated OTP, but it also guarantees a trusted path with the user when it is displayed. TrustOTP implements multiple OTP algorithms for different application scenarios and standards.

7.10.5 NFC

NFC services often require a high level of security. The TEE are used in some related work in order to enhance their global security. In [ALZR13], authors propose to implement the reader firmware inside the TEE. Hence, NFC-enabled smartphones can be used as card readers with the same security level as standalone card readers.

[Ste13] shows that the SE access control mechanisms defined by GlobalPlatform are flawed. Indeed, SE (Secure Element) access is granted only to some applications. The authors demonstrate the shortcomings of the existing solutions and propose an alternate model to prevent an illegal access to the SE.

7.10.6 Trusted Sensors

Many mobile applications rely on sensors embedded in smartphone devices. Some applications may need inputs from sensors such as fingerprint reader, GPS, or camera in order to perform certain tasks. For instance, Apple Pay uses Touch ID, a biometric fingerprint authentication technology to validate transactions. Other mobile payment applications use location as a second factor authentication scheme [MKS+14]. In fact, secure applications often require authenticity and integrity of sensor readings. Therefore, the need for "trusted sensors" is recognized.

Liu et al. in [LSWR12] propose two programming abstractions to support trusted sensors: sensor attestation and sensor seal. They leverage the TEE to protect the integrity of sensor readings. The TEE trusted I/O path property simplifies the design of a small stack of trusted sensors. Authors also address the privacy concerns raised by trusted sensors. They anonymize the readings of a GPS sensor using the techniques of differential privacy.

7.10.7 Trusted Platform Module

Trusted platform modules (TPMs) are designed to be deployed with personal computers. Recent efforts were made in order to adapt TPMs to mobiles and embedded systems. However, most of resource-constrained systems cannot bear an additional chip on their motherboards for reasons of cost and power consumption. Therefore, and in contrast to common TPMs, TPMs for mobiles and embedded systems should not be implemented as separate chips. Instead, they should be implemented in software.

The TEE offers a hybrid approach somewhere in the middle between a dedicated TPM chip and a pure software TPM implementation. It enables the implementation of TPM on a software-only basis, without the need for additional special purpose hardware. The TEE provides enough security to protect the execution of the TPM emulator.

There exist several implementations of TEE-based TPM in the literature [DW09, DW10, RSW+16]. Raj et al. in [RSW+16] describe a typical architecture that enables the use of TEE to provide secure code execution isolation without a hardware TPM. [RSW+16] defines the reference implementation used in almost all mobile devices running Windows, including Microsoft Surface and Windows Phone.

In [DW09], authors present two approaches to implement TPM in embedded systems. The first approach is to use a smart card. The TPM is implemented as a JavaCard applet. The second approach is to run the TPM in a TEE. The authors present a comparison study between the two approaches. However, they do not make a definitive statement on the difference between the security properties of the JavaCard-based TPM and the TEE-based TPM.

[DW10] defines a flexible, modular and customizable architecture for the mobile version of the TPM, namely Mobile Trusted Module (MTM). Two concepts for dynamic command loading are discussed. The implementation is based on Java and takes advantage of the protection facilities provided by the Java virtual machine (JVM). The JVM avoids most of the common pitfalls of low-level languages due to the design of its memory model.

7.10.8 Self-Protection

TEE was used recently to provide self-protection for autonomic systems. Azab et al. perform real-time protection for kernels of mobile devices [ANS+14], while authors of [GJ14] propose introspection mechanisms for operating systems using TEE. SKEE (for Secure Kernel Execution Environment) [ASB+16] is designed to help kernels in mitigating *code-injection attack*. This attack consists in deceiving the victim kernel into executing malicious code by placing it in a kernel memory region, thereby corrupting some function pointers. SKEE prevents this attack by protecting the integrity of the kernel state and by preventing the kernel from performing

some high privileged instructions. SKEE can be perceived as a highly efficient micro-kernel that handles the security of the main kernel.

7.11 Attacks on TEE

TEE has been heavily promoted as the silver bullet solution that provides secure processing in mobiles and embedded systems. However, far from speculative bubbles and marketing claims, security experts have not put TEE to the test, especially because of non-disclosure agreement (NDA). The attack surface of TrustZone-based TEE is: software exceptions (e.g. SMC call), hardware exceptions (e.g. interrupts), shared memory interface, peripherals, and TEE-specific calls. The threat model includes a powerful attacker who is able to execute an arbitrary code in the kernel privileges of the normal OS.

To the best of our knowledge, four attacks have been published against the Qualcomm TEE (i.e. QSEE) or a manufacturer-customized version of it. QSEE is an enticing target for attackers, since Qualcomm controls the majority of the market of Android devices. In addition, it is easier to exploit security flaws, as the memory layout of QSEE is known. In fact, the QSEE resides unencrypted on eMMC flash and loaded at known physical address. Disassemblers are used to gain insight into QSEE implementation. In [KH14], authors present an exploit that is caused by code added by HTC. The exploit enables the execution of an arbitrary code within TrustZone in the secure region of the memory. Rosenberg unlocks the bootloader of Motorola Android phones using two different exploits. The first exploit is about overwriting part of the secure region of the memory with certain values [Ros13]. This is used to bypass the check of the function that unlocks the bootloader. The exploit works only on Qualcomm-based Motorola Android phones. The second exploit affects all Android phones that utilize Qualcomm Snapdragon SoC [Ros14]. By issuing specially crafted SMC requests, an attacker can execute an arbitrary code inside the QSEE. This vulnerability may be used to compromise any applications relying on TEE for security. Laginimaineb in his (or her) blog has shown that the QSEE kernel does not perform any validation on the supplied arguments, notably it freely uses any given pointer [Lag16]. This can be used to read or write any value at any address on the secure memory. Laginimaineb demonstrates the exploit by executing an arbitrary code inside the TEE kernel. This allows attackers to hijack any QSEE application, thereby exposing all its internal secrets. For instance, Laginimaineb leverages the exploit to reveal the master TEE key used to encrypt the Android file systems.

Besides exploiting SMC calls, the shared memory can be manipulated to find vulnerabilities. In [Sen13], a module intercepting exchanged data between the normal world and the secure world is implemented. This module is integrated inside the kernel driver which interacts with the TEE. The targeted TEE by this attack is MobiCore and its goal is to better understand the internal workings of MobiCore trustlets.

7.12 Conclusion

TEE has practical interests and can be used to construct complex systems. Thus, we believe that deeper understanding of TEE is required in order to design better and more trustworthy systems. The relevance of our theoretical approach will be shown in next chapter.

In this chapter, we proposed a refined definition of TEE to contribute in establishing a common understanding of this term in the context of trusted computing. Furthermore, we examined the building blocks of TEE explicitly differentiating it conceptually from the classical notions of SEE and DRTM. Moreover, we discussed how trust can be measured as well as formal verification for TEE. Finally, we discussed TrustZone-based TEE, by providing a short survey using our proposed definition, presenting the known attacks and classifying the proposed applications of TEE in the

literature. As we have seen, the variety of the existing solutions confirms the great potential of this technology. In addition, the recent disclosed attacks against some industrial TEE show its omnipresence in the market of smartphones. It might seem peculiar to make such affirmation. We argue that only dead technologies are not attacked. Attackers are apt to invest time and money in exploiting technologies only if they are widely deployed in real systems.

Chapter 8

Reconsidering the Power of Trusted Execution Environment

"Don't tell me you already found the solution!
Wondered Agasa while staring at Conan's face.
Not at all, replied Conan with a wild smirk,
but the less I understand the more interesting it becomes!"

– Gosho Aoyama, Detective Conan, Professor's Treasure Box

Contents

8.1 Trusted Execution Environment Based on Garbled Circuits 138
8.1.1 Introduction 138
8.1.2 Background 139
8.1.3 Two-Party Secure Computation 139
8.1.4 Motivation and Threat Model 141
8.1.5 TrustedGarble Design 142
8.1.6 Development in TrustedGarble 144
8.1.7 Security Analysis 145
8.1.8 Open Issues 145
8.1.9 Related Work 146
8.1.10 Conclusion 147
8.2 Towards Integrating TEE Into Autonomic Systems 148
8.2.1 Introduction 148
8.2.2 Autonomic Computing 148
8.2.3 Self-Protection 149
8.2.4 Self-Healing 149
8.2.5 Problem Statement 149
8.2.6 Threat Model and Assumptions 150
8.2.7 Architecture 151
8.2.8 Open Issues 153
8.2.9 Conclusion 154

The Trusted Execution Environment (TEE) was designed to define an isolated processing environment in which arbitrary code can be securely executed. Compared to the tamper-resistant Secure Element, TEE is commonly perceived as a balance technology that provides more computation power, but ensures less security. In this chapter, we investigate two new research directions

in trusted computing. Firstly, we propose TrustedGarble, a TEE that protects its sensitive data even when its secure kernel gets compromised. To this end, we make use of ARM TrustZone and leverage efficient techniques for evaluating garbled circuits in the semi-honest model. The used techniques only require 10 ms to encrypt 16 bytes using AES with 128-bit key. TrustedGarble allows service providers to securely execute their applications without having to fully trust the underlying TEE. Secondly, we apply the TEE principles in another context that is different from smartphones, namely embedded autonomic systems. We rely on the strong TEE isolation to guarantee two important properties of autonomic systems: self-protection and self-healing. Part of the findings of this chapter was published in [SAB15c] that was presented at the 12th IEEE International Conference on Autonomic Computing (ICAC 2015).

8.1 Trusted Execution Environment Based on Garbled Circuits

8.1.1 Introduction

Trusted Execution Environment (TEE) is an isolated processing environment that runs on a separation kernel following the dual-execution-environment approach (refer to the previous chapter for more details). It protects the integrity and the confidentiality of its running applications that, hereafter, we call trustlets. It is increasingly deployed into commercial smartphones [And16e, ANS+14, Seq16a, Tru16], and Android N makes TEE-based architectures mandatory for all devices [Arm16b]. The term TEE is widely used in advertisements of platform providers for marketing purposes [Qua16a, Sol16].

Nevertheless, many service providers are still reluctant to adopt this technology. The reason behind this is twofold. First, the TEE is often perceived as a tradeoff between security and computation power [Glo16b]. However, some argue that such a tradeoff is not necessary, since numerous complex architectures can be heavily optimized to work efficiently on Secure Elements (see [CPST15] for a practical E-Cash solution). Second, the current TEE execution model requires that the TEE has full access to its trustlets binaries. This model raises several problems, especially when trustlets include static secret keys. Indeed, it means that these keys could be recovered by an attacker when the TEE is compromised. In addition, it also means that service providers must have total trust that during the execution of trustlets, the TEE will not attempt to steal their data to sell them out to third parties. This problem is inherent to the TEE design of GlobalPlatform [Glo11b], and therefore it concerns all related architectures.

Previous Attempts

To improve this situation, a body of research studies has investigated the design of alternative execution models that provide better protection. These models allow TEE to properly execute trustlets without having access to their sensitive data. Related work mostly delegates the operations manipulating such data to another trusted entity, like Secure Element [AFA+13] or Cloud [UA16]. However, existing approaches fall short to build practical systems. On the one hand, the solutions relying on Secure Element generally split trustlets into two parts: one installed on TEE and the other installed on Secure Element. This approach suffers from several shortcomings concerning its scalability and its deployment cost. As a matter of fact, Secure Elements have limited resources [RE10] and cannot host one application for each trustlet. In addition, Secure Elements are strongly controlled by their owners, and thus installing new applications costs money and makes the whole architecture more complex to manage. On the other hand, the Cloud based approach raises serious issues related to users privacy. Indeed, this solution requires users to upload their inputs to a remote Cloud server in order to perform the required operations. Therefore, a curious server can build precise profiles about each user, which could cause some privacy threat.

Overview of TrustedGarble

This chapter presents **TrustedGarble**, a full-fledged TEE that raises the bar of trustlet protection without adding a trusted entity. TrustedGarble achieves the same level of security required to host highly sensitive applications. To this end, it solves multiple technical challenges. First of all, TrustedGarble has to protect the trustlets secret data from the TEE kernel. For instance, if an attacker succeeds in compromising the TEE kernel, it must not be able to compromise the secret data of its trustlets. This isolation is non-trivial to achieve given that the TEE kernel is required to run at a higher privilege level than the one of trustlets. Second, TrustedGarble is required to preserve users privacy. Hence, trustlets execution must be possible without having to send users data to a remote server. Finally, TrustedGarble is required to execute the entire trustlet inside the TEE kernel, thereby not including any additional entity. We notice that each requirement is individually not hard to accomplish, but achieving them together is a real challenge, since it implies that the TEE kernel can *magically* make secure computations even though it has no access to the needed sensitive data. For example, TrustedGarble is required to be able to execute a trustlet that signs some data and still learns nothing about the secret key used for the signature.

Summary of Challenges

The challenges that TrustedGarble tries to solve can be summarized as follows:

1. It builds a unique solution that offers better protection, hence ending with the tradeoff between security and computation power for TEE applications;
2. It defines a more flexible execution model where service providers need only to partially trust the TEE security as well as its owner;
3. It provides a complete design of the TrustedGarble architecture describing its different components. To offer its guarantees, TrustedGarble leverages efficient techniques of secure two party computation in the offline/online setting. In particular, it relies upon building component-based garbled circuits.

8.1.2 Background

In order to better understand this section, readers are required to be familiar with some background information:

- Smartphone security and its security challenges (section 2.2);
- Secure Element (section 2.4.2);
- ARM TrustZone (section 2.4.3);
- The dual-execution-environment approach (chapter 6);
- Trusted Execution Environment (chapter 7).

8.1.3 Two-Party Secure Computation

Oblivious Transfer

Oblivious Transfer (OT) is a fundamental building block for secure computation [Rab81]. It is a cryptographic protocol executed between a sender $\mathcal{S}$ and a receiver $\mathcal{R}$, in which $\mathcal{R}$ *obliviously* selects one of the inputs provided by $\mathcal{S}$. More specifically, in 1-out-of-2 OT, the sender $\mathcal{S}$ provides

two messages (m_0, m_1). OT ensures that the receiver $\mathcal{R}$ succeeds in obtaining m_c, where $c \in \{0, 1\}$, while learning nothing about m_{1-c} and without revealing the chosen message to $\mathcal{S}$. This simple case can be generalized for n bits. Indeed, the sender $\mathcal{S}$ provides $\mathcal{L}$, a list of n 2-tuples. Then, the receiver $\mathcal{R}$ succeeds in obliviously obtaining n elements according to some n-bit string X as follows: the ith element is received by successively applying the simple OT on c^i and (m_0^i, m_1^i), where c^i is the ith bit of X and (m_0^i, m_1^i) is the i-th 2-tuple of $\mathcal{L}$.

In literature, there are many proposed protocols performing OT, most notably [NP01], which require public-key cryptography, and therefore are inefficient in real-world systems. OT extension [IKNP03] is a technique reducing the number of expensive public-key operations to instead use faster symmetric cryptographic operations. Recently, the efficiency of OT extension protocols has gained a lot of attention. Several optimizations were proposed in [ALSZ13, KK13] that decrease by half the amount of communicated data between the sender and the receiver. This builds faster secure computation schemes.

Garbled Circuits

While Yao's [Yao82, Yao86] are often cited as a reference for Garbled Circuits, this term was coined by Beaver et al. in [BMR90]. A Garbled Circuit (GC) is a cryptographic tool that allows one to evaluate encrypted functions on encrypted inputs. The evaluated function must be represented as a boolean circuit [Vol99].

More formally, Bellare et al. [BHR12] define a garbling algorithm Gb as a probabilistic algorithm that transforms a function $f : \{0, 1\}^n \longrightarrow \{0, 1\}^m$ into a triple of functions $(F, e, d) \longleftarrow \mathsf{Gb}(f)$, where:

- the *encoding function* e turns an input $x \in \{0, 1\}^n$ of the function f into a garbled input X using a secret key K. We write $X = e(\mathsf{k}, x)$.

- the *decoding function* d turns a garbled output Y into the corresponding output $y \in \{0, 1\}^m$. We write $y = d(Y)$.

- the *garbled function* F takes a garbled input X and gives a garbled output Y. We write $Y = F(X)$.

Given a function f, some input x and a garbling algorithm Gb, a garbling scheme works as follows: Gb is first applied on f to generate (e, F, d). Then, the garbled input X is obtained using the encoding function e. The final output is computed via $y = d(F(X))$. We require that $f(x) = d(F(X))$. A garbling algorithm should satisfy several security properties. The most important one is *privacy*, since this property is essential for secure function evaluation. Here, privacy means that an adversary acquiring (F, X, d) can learn no more that what is revealed by knowing just the final output $y = d(F(X))$. Formal definitions and security proofs are given in [BHR12].

Because of its extensive computational cost, the GC approach has traditionally been considered more of a theoretical curiosity than a practical mechanism to build secure function evaluation solutions. However, today, we are witnessing a surge of the theoretical and implementation techniques that have improved the efficiency and the practicability of the GC protocol, see [BHKR13, MGBF14, MGC+16, SHS+15, ZRE15]. The technique of GC has been extremely influential, engendering an enormous number of applications and implementations. Indeed, many security-critical applications have started to integrate practical realization of GC into their architecture. This includes secure search, packet inspection of encrypted traffic, biometrics matching and private set intersection [EHKM11, HEK12, SLPR15].

Two-Party Secure Function Evaluation

Secure Function Evaluation (SFE) has been introduced in the seminal work [Yao82, Yao86]. Loosely speaking, two-party SFE enables two parties to jointly evaluate an arbitrary function over their respective private inputs in a way that the evaluation reveals only the output of the function. Henceforward, unless stated otherwise, we use the term SFE to only denote GC-based two-party SFE. Here, the construction of SFE involves two parties: generator and evaluator.

More formally, a SFE scheme can be built by a modular combination of an arbitrary garbling algorithm with an arbitrary OT protocol. In SFE, the generator has a public function f. Both the generator and evaluator have their own private input x, x'. SFE requires that at the end the generator learns nothing, while the evaluator learns no more than the output $f(x, x')$. In a nutshell, given a function f and a garbling algorithm Gb, the generator starts by garbling (i.e. encrypting) the Boolean circuit of the function f and obtains (e, F, d). It also garbles (i.e. encodes) its own inputs $X = e(x)$ to send them, together with the garbled F and the decoding function d to the evaluator. Then, the evaluator obtains the garbled version of its inputs $X = e(x')$ through OT and subsequently it evaluates $y = d(F(X, X'))$. Thus, the generator never gets to know x' and the evaluator gets the output of $f(x; x')$ while learning nothing about x. The security proofs of SFE are given in [BHR12, LP09].

Adversaries Model

Depending on their power, there are two types of SFE adversaries:

- *semi-honest* (or passive) adversaries faithfully follow the protocol specification, but attempt to learn additional information by analyzing the exchanged messages during the execution [LP09].
- *malicious* (or active) adversaries can arbitrarily deviate from the protocol specification in order to learn additional information.

Despite their weak security guarantee, we only consider semi-honest SFE protocols due to the fact that SFE protocols in the powerful malicious threat model are too inefficient. Protocols in the semi-honest model is relevant in two cases: (1) when parties are legitimately trusted, but are prevented from divulging information for legal reasons, or want to protect against future compromises; and (2) where it would be difficult for parties to change the software without being detected, either because software attestation is used or due to internal controls in place.

8.1.4 Motivation and Threat Model

Motivation. We begin this section by underlining our motivation to build a trusted execution environment using the paradigm of secure computation. First of all, TEE is often presented as a balance technology that offers more powerful processing capabilities than Secure Element (SE), but guarantees less security. Indeed, it has been certified at EAL2 by Common Criteria [Glo15b], while technologies related to SE are certified at EAL4+ [Ora12] (as a matter of fact, EAL1 is the lowest evaluation level and EAL7 is the highest one). Moreover, SE is designed to withstand physical attacks, while the ARM TrustZone (i.e. the TEE underlying hardware) does not assure high levels of tamper resistance and physical security [ARM09]. Therefore, many service providers express their concerns about the effective security of TEE. Their concerns have increased recently because of several disclosed attacks breaching the core security of some widely deployed TEE systems [KH14, Lag16, Ros13, Ros14].

In addition to inadequate protection, service providers also worry about the TEE trust model. In fact, the current model gives the TEE owners so much power that they are capable of compromising trustlets at runtime. This could have severe consequences. For instance, they might

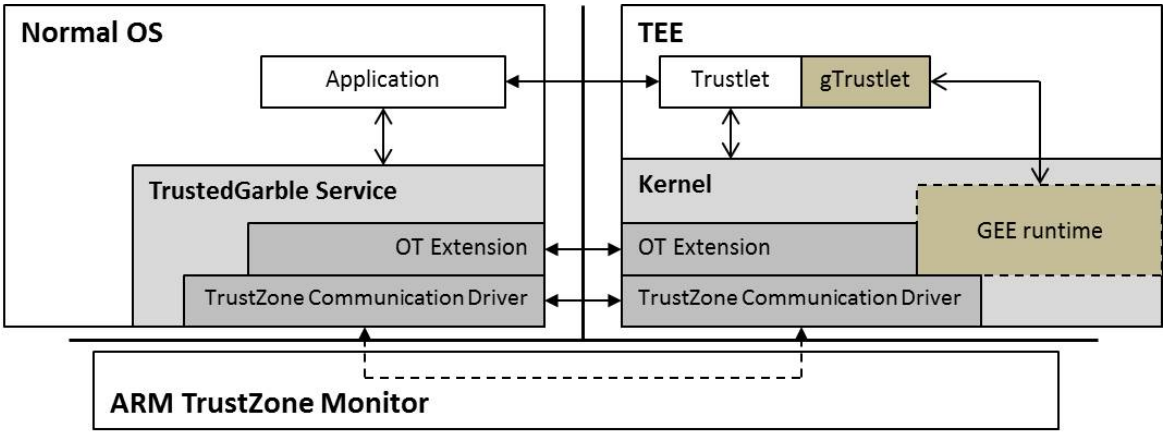


Figure 8.1: TrustedGarble Architecture

be coerced to comply with law enforcement to give out some sensitive data related to a specific trustlet. Such a case does not exclude the possibility that the owner sells these data to other parties. Consequently, some may cast doubt on the trustworthiness of TEE, since its security guarantees, especially the confidentiality of sensitive data, depend also on the good intentions of its owner.

In light of the above observations, we propose TrustedGarble, a TEE protecting the sensitive data of its trustlets even if its kernel is broken by an attacker, including its owner.

Threat Model. TrustedGarble protects its trustlets against the following threat model. The attacker is assumed to successfully execute arbitrary code inside the TEE kernel. The attacker is also assumed to be able to eavesdrop any communication interface with the TEE. However, we do not consider side-channel attacks nor physical attacks that fall outside the defense capabilities of ARM TrustZone [ARM09].

Furthermore, we assume that both the TEE owner and the trustlet provider behave semi-honestly, i.e. they follow the SFE protocol, but they may try to infer additional information about the other party's input. The semi-honest model is suitable for TEE, since code attestation and secure boot can be applied (refer to chapter 7). Moreover, this model performs better than the one including two mutually-distrusted parties.

Security Guarantees. TrustedGarble provides two security guarantees. First, it prohibits the TEE kernel from having access to the trustlets sensitive data. As a result, even if an attacker completely compromises the TEE kernel, it would not be able to obtain any sensitive data. Second, TrustedGarble does not send the users' inputs to the service provider in order to execute its tasks. As a result, it protects users privacy from curious service providers. For example, TrustedGarble can ask a trustlet to sign some data, while never loading the required private key in its memory and without communicating the data to the service provider.

8.1.5 TrustedGarble Design

TrustedGarble is based on ARM TrustZone. This technology allows systems to efficiently provide two isolated execution environments: an untrusted one where the main OS (also called Normal OS) run, and a trusted one where the TEE runs (refer to chapter 6). With TrustedGarble, a secure application is partitioned into two components: a small-sized trusted component called **trustlet** and executed by the TEE, and a large-sized untrusted component that implements most application functionalities. To enable interaction, there should exist a communication channel between the two isolated environments. This partitioning process is similar in spirit to previous

work on TEE (refer to chapter 7).

The particularity of TrustedGarble is that trustlets themselves are divided into two parts. Indeed, the code requiring sensitive data (e.g. secret keys) is isolated before being encrypted using GC techniques. This part, called *garbled trustlet* **(gTrustlet)**, is then 'glued' to the trustlet via intermediate proxies. During runtime, trustlets are executed directly by the TEE kernel, while the evaluation of their corresponding gTrustlets is managed by a special component called *Garbled Execution Environment* **(GEE)**.

Figure 8.1 illustrates the high-level design of TrustedGarble. To meet our goals, we provide five primitives:

1. TrustedGarble Service. TrustedGarble Service proposes a high-level communication interface between the TEE and the Normal OS. It bridges the trustlet requests coming from the untrusted application code and TrustedGarble. The actual communication is managed by the *TrustZone Communication Driver* that leverages the ARM TrustZone Monitor to establish a channel between the two isolated worlds. This driver is based on a set of low-level TrustZone mechanisms that are responsible for world switching, interrupt handling, communication and protection. It is designed to satisfy the requirements of inter-environment communication as they are mentioned in section 7.4 of chapter 7.

2. Trustlet. A trustlet is an application that runs inside the TEE kernel. It specifies an interface defining the functionalities provided to the untrusted application. The main part of trustlets is written in a small set of Java and runs in a special Java Virtual Machine. The part requiring to manipulate sensitive data is identified and isolated from the rest. It is written in a custom C-style language, compiled into boolean circuits and then encrypted using the GC protocol. This part is called gTrustlet and it runs inside the GEE Runtime Environment. The reason of using a new language is to produce efficient representations of boolean circuits [MGC$^+$16]. Because of the special nature of gTrustlets, the process of writing trustlets might seem cumbersome. However, we note that developers need not bothering with all the technical details behind trustlets. The building tool transparently garbles gTrustlets and generates all the necessary objects to make secure procedure calls between the garbled part and the rest of the trustlet.

3. GEE Runtime. GEE is a custom interpreter for garbled circuits. It relies on the component-based GC technique in which garbled circuits can be constructed in a modular way. Indeed, GEE comprises of pre-garbled libraries performing common tasks, such as AES encryption. Thus, gTrustlets mainly consist of chaining these pre-garbled libraries together in order to define their function. They might also define a narrow set of garbled procedures that are specifically tailored for themselves. This technique, explored and proved efficient in [GLMY16], reduces the amount of data that must be communicated during the GC evaluation, and therefore greatly improves the computation time of trustlets. For instance, it can perform AES encryption of one block of data (i.e. 16 bytes) in only 10 ms. It can also encrypt 160 bytes of data using AES in CBC mode in 450 ms. In order to increase the GEE performance, we leverage the dedicated cryptographic module that is integrated into modern smartphones and which provides hardware acceleration for commonly used algorithms such as AES [Ele15].

4. OT Extension. As its name indicates, this primitive handles the necessary operations of any required OT. OT Extension is logically split into two parts. This is due to the fact that TEE kernel does not support network operations, since simplicity is one of its design objectives. Thus, the part running in the Normal OS communicates directly with the service provider to receive the required data, while the one in the TEE parses the OT requests from the GEE and encodes them before forwarding them to the Normal OS.

5. Secure Boot and Remote Attestation. TrustedGarble assumes that GEE (and the whole system) is loaded securely. Hence, the TEE kernel is setup securely during boot up time. This process is straightforward using Secure Boot. It is worth noting that Secure Boot guarantees the integrity/authenticity of the GEE only during the boot. Indeed, it cannot guarantee the integrity of the GEE after the TEE runs. Therefore, Remote Attestation is used in order to verify that the GEE has not been maliciously tampered with. We require that the service provider ask for a Remote Attestation for each evaluation of gTrustlet. This allows TrustedGarble to adopt the semi-honest model instead of the malicious one, thereby gaining in terms of efficiency.

8.1.6 Development in TrustedGarble

Typical Development Scenario

To build a secure mobile application with TrustedGarble, a developer typically performs the following three steps:

1. Determine which part of the secure application makes the trustlet.
To define the trustlet, the developer first determines the program logic that needs to operate isolated from the main OS. Then, the developer carefully defines the public interface to the trustlet, since this interface controls what functions can be called from the Normal OS using TrustedGarble Service. Finally, the main core of the trustlet is written in Java.

2. Isolate the trustlet part handling sensitive data.
The developer identifies the trustlet sensitive data and separates all code manipulating these data. Then, the developer defines a new class declaring a list of prototypes of native Java functions using JNI (Java Native Interface). These functions are then implemented in a custom C-style language to constitute the gTrustlet. The gTrustlet is mainly primarily defined by calling the pre-garbled library that is already loaded into GEE. Finally, the building tool of trustlets compiles the Java code, garbles the gTrustlet and transparently binds the two components to make one logical program. This program can be then executed inside the TEE kernel that delegates the evaluation of the gTrustlet to GEE.

3. Deploy trustlet and keep garbled copies of its gTrustlet.
At this point, TrustedGarble loads the trustlet and creates an instance of its main class. The resulting object constitutes the runtime state of the trustlet until it is destroyed. To allow an application in the Normal OS to interact with the trustlet, the application requests that TrustedGarble Service creates a special entry-point object. This object is the communication interface to the trustlet. Indeed, whenever the application calls methods on the entry-point, it forwards them to the trustlet main object. In case a call requires a function in the gTrustlet, a network channel is set with the corresponding service provider and the execution is switched to GEE. Once the execution is finished, another garbled copy of the gTrustlet is sent by the service provider. This is because, unfortunately, garbled circuits are no longer secure if they are used more than once [GKP+13]. The goal of this final operation is to optimize the computation time of the next gTrustlet evaluation.

Parties Interaction

There are four actors in the TrustedGarble ecosystem: (1) A *trusted entity* that implements GEE and integrates the appropriate attestation mechanisms asserting its integrity/authenticity inside the root of trust; (2) The *TEE owner* that designs the TEE kernel and all its related development

environment; (3) A *Service Provider* that requires to install a secure application inside TrustedGarble; and (4) The *User* that needs her smartphone to execute the secure application of the Service Provider. We note that the Trusted Entity and the TEE owner must be two independent entities.

Here, we define how these four actors interact with each other, so that TrustedGarble accomplishes its tasks.

- The TEE owner collaborates with the Trusted Entity in order to integrate the GEE inside TrustedGarble.
- The User installs the secure application developed by the Service Provider.
- When the User requires to execute her application, a secure communication must be established between the TEE and the Service Provider in order to perform the necessary OT messages for the secure function evaluation. The network connection is also used to send a Remote Attestation that attests the integrity/authenticity of the GEE.
- At the end of the trustlet execution, the trustlet discards its gTrustlet, since using a gTrustlet twice would compromise its security.
- The Service Provider pushes a new gTrustlet to the User smartphone. This operation is for optimizing the execution time, and thus it can wait until the User has a decent Internet connection.

8.1.7 Security Analysis

Following the TEE core design, the attack surface of TrustedGarble comprises that of a TEE. This means that, please refer to the TEE definition in section 7.3.2, TrustedGarble protects against all software attacks that come from the Normal OS. This is mainly due to the fact that TrustedGarble leverages ARM TrustZone for its isolated execution. Indeed, ARM TrustZone controls the interaction between the Normal OS and the TEE via a special interface using the newly introduced SMC instructions (refer to section 2.4.3 in chapter 2) [ARM09]. The communication interface is quite small, which limits the exposure of code vulnerabilities.

Beyond such a strong security guarantee, TrustedGarble provides a complete protection against physical attacks. As a matter of fact, an attacker with the capability of tampering with the hardware can disable the TrustZone protection. Moreover, an attacker can also exploit a bug in the TrustedGarble kernel by carefully injecting crafted code sequence in the trustlet code. However, and despite being powerful, these attacks cannot succeed against our design. Indeed, the maliciously injected code would try to read or to overwrite the sensitive data of a trustlet. For instance, it would attempt to reveal the private key used for signature generation. Nevertheless, the malicious code cannot find such data, since they are not embedded inside the trustlet binaries. In addition, the SFE scheme ensures that confidentiality of the inputs (i.e. sensitive data) used by the Service Provider during the execution of the trustlet. This implies that the GEE has not been tampered with, since for efficiency reasons we decided to take the semi-honest model for our SFE scheme. We protect the security of GEE via two mechanisms. First, the Secure Boot attests its integrity during the boot of the smartphone system. Second, the Remote Attestation that is necessary before each trustlet execution ensures that the GEE is still protected during runtime. Consequently, the sensitive data are protected even of an attacker breaches the security of the TEE kernel.

8.1.8 Open Issues

Here, we summarize some of the design challenges that we have not yet addressed in this thesis.

Reduced Storage Overhead

Our use of component-based GC offers significant benefits in terms of efficiency and computation time, but it comes at a noticeable cost to storage size. Indeed, each pre-garbled component implements one specific function, and therefore this approach requires storing a large set of components in order to offer a rich API. This means that the size of GEE might become prohibitive for mobile devices in case many basic functions are needed. Thus, the storage size of TrustedGarble is in conflict with its goal of providing a realistic and representative set of functionalities. Finding the correct optimization and balance between storage size and the number of pre-garbled components is a challenging issue that GEE shares with any component-based GC system.

Ease of Programming

Our current design of TrustedGarble does not offer a tight integration with the Android framework. Such lack of integration results in a poor level of programming abstractions, and consequently this makes TrustedGarble less developer-friendly. To support this and so gain in productivity, we require to design a building tool that allows developers to simply define Java classes and then use RMI (Remote Method Invokation) techniques to call the secure part in their corresponding trustlet. We also require that generating all the necessary proxies for inter-world communication (i.e. between Android and TrustedGarble) be transparent for developers. An inspiring work in this domain is [SRSW14] that provides a rich development experience for TEE.

Formally Verified Interpreter for GC

Due to the critical nature of the GC system component in our architecture, the security of TrustedGarble could be compromised if the underlying GEE produces erroneous results. Authors in [MGC+16] show that the existing GC frameworks are unreliable by demonstrating their unpredictable behavior in many cases. Therefore, the authors emphasize that there is a need to GC frameworks whose correctness was validated through principled design, but for practicality issues, they rely on 'validation by testing' to build their framework. One area for future research is to apply formal verification tools to produce a formally *verified implementation* of GEE.

Complete Proof-of-Concept

We admit that this work lacks of a full proof-of-concept that shows the feasibility of our architecture. However, considering all the existing open-source projects on TEE and GC, we are highly confident that a complete implementation would only confirm the relevance of our approach. We believe that a theoretical overview of TrustedGarble, a GC-based TEE, has interest of its own. Indeed, our study will remain valid independently from the current performance of the GC techniques and the computation power of smartphones.

8.1.9 Related Work

TrustedGarble borrows many concepts from previous work on building trusted execution environments. It adopts two main ideas that are used in trusted and secure computation:

1. Relying on ARM TrustZone technology to ensure security by isolation for the secure applications [ARM09];

2. Using garbled circuits to perform secure function evaluation [Yao82, Yao86].

Nevertheless, there are key differences between our work and existing approaches. First, TrustedGarble protects the sensitive data of trustlets beyond the security of its TEE kernel.

Generally, the threat model of TrustZone-based TEE, such as TLR [SRSW14] and GlobalPlatform architecture [Glo11b], does not suppose that the TEE kernel could be compromised by an attacker, and therefore their security guarantees do not hold once the kernel gets compromised. Second, TrustedGarble improves the computation time of garbled circuits by leveraging some of the TEE core aspects. This is due to the fact that Secure Boot and Remote Attestation allow our component-based GC system to run in the semi-honest model which performs better than the malicious model. Indeed, Secure Boot ensures that GEE will boot into a secure state and the process of Remote Attestation generates unforgeable proofs of this state for each private evaluation. This allows service providers to be sure that they do not interact with a malicious GEE component which attempts to cheat in order to breach the security of the GC protocol.

As for secure two-party computation on mobile devices, the closest piece of work to TrustedGarble is [DSZ14]. Authors develop a generic secure computation on Android smartphones by using the Goldreich-Micali-Wigderson (GMW) protocol. The particularity of this work is the integration of Secure Element in the execution of the protocol, which allows several optimization in their scheme. TrustedGarble's primary benefit over [DSZ14] is the use of TrustZone that, compared to Secure Element, provides more powerful computation power. This allows TrustedGarble to perform much faster in some scenarios (see section 6.2 in [DSZ14]). Moreover, TrustedGarble improves the installation (aka setup) time of secure applications. We recall that our component-based GC system substantially reduces the communication overhead of installation, since it only requires to link the pre-garbled components together to specify the required function. Therefore, it performs better and also saves users cellular data.

8.1.10 Conclusion

We introduced TrustedGarble, an environment for running trusted applications on smartphones. TrustedGarble offers strong security guarantees for trustlets sensitive data. With TrustedGarble, programmers can write trustlets and specify which parts of the trustlet should be strongly protected. These parts, called garbled functions, remain secure even when the TEE kernel is compromised.

TrustedGarble uses garbled circuits, a method originally proposed by Yao to allow two parties to jointly evaluate a function without learning the inputs of the other party. The rich hardware support of ARM TrustZone combined with the efficiency of the approach of component-based GC allows TrustedGarble to offer a secure, yet rich programming environment for developing trusted mobile applications. In addition to presenting the design of TrustedGarble, we have also discussed some of the open issues related to our architecture. Our primary goal here is to ignite the discussion about new directions in trusted computing on mobile devices.

8.2 Towards Integrating TEE Into Autonomic Systems

8.2.1 Introduction

An embedded autonomic system is designed to be trustworthy in order to avoid failure despite the hostile conditions of their deployment environments. Trustworthiness is an important aspect of self-protection, which is one of the main properties of autonomic computing. It means that an autonomic system must protect itself from malicious attacks from both system user and malicious parties. However, self-protection alone does not ensure trustworthiness. This explains the recent trend to integrate trusted computing concepts into embedded autonomic systems. Cloud computing is also concerned by this trend.

Some embedded systems are designed to be reliable, in the sense that they should keep on working without human intervention. With the advent of autonomic computing, reliability is achieved by self-protection and self-healing. Very often, embedded autonomic systems include specific modules that provide the properties of self-protection and self-healing. Such modules are themselves vulnerable, and hence need to be protected. In this work, we propose to shield these modules inside a trusted execution environment. The protection is achieved by securing the embedded system kernel using two independent mechanisms: introspection, and "trap and emulate".

8.2.2 Autonomic Computing

Autonomic computing aims at making computing systems self-managed, thereby simplifying their deployment by decreasing human involvement. The term "autonomic" is inspired from biology. Indeed, the human body is able to adapt to external and unexpected changes in the surroundings environment. To this end, the nervous system is able to effectively monitor, control and regulate the human body without external intervention (except of course when serious health issues rise).

Autonomic computing attempts to follow the same principles as its biological counterpart. It was IBM that launched the autonomic computing initiative in 2001 to describe systems that are said to be self-managing [IBM01]. They were motivated by solving the 'grand challenge' that was facing the IT industry at that time, namely its growing complexity. They argued that self-managed systems are the means of overcoming this challenge.

Since then, *autonomic computing* aims at making computing systems self-managed. On other words, its objective is to enable computer systems to manage themselves, so as to minimize the need for human intervention. There are four fundamental principles for autonomic computing systems to accomplish its goal [HM08, WHW$^+$04]:

1. *Self-Configuration:* the system sets and controls its internal parameters so as to adapt to dynamic changes;

2. *Self-Healing:* the system detects and repairs failed components so as to maximize its availability;

3. *Self-Optimization:* the system proactively strives to optimize its operation so as to improve the efficiency with respect to predefined goals;

4. *Self-Protection:* the system anticipates, identifies and prevents various types of threats in order to preserve its integrity and security.

These fundamental principles are undoubtedly interdependent. For instance, a breach through the self-protection principle may trigger the self-healing mechanisms to reset the system to its original safe state. In this work, we focus solely on the self-healing and the self-protection principles that we describe in more detail below.

8.2.3 Self-Protection

An autonomic system shall anticipate, detect, identify and protect itself from internal and external threats. This aims at achieving security, privacy and data protection. Self-protection addresses various threats, including malicious attacks or accidental dysfunctions. In addition to reactive self-protection, an autonomic system shall proactively anticipate security threats to prevent their occurrence. Similar to a biological immune system, autonomic systems should protect themselves while requiring minimum or zero user intervention.

8.2.4 Self-Healing

An autonomic system shall detect, diagnose and recover from malfunctions while trying to minimize service disruption. Moreover, an autonomic system shall predict potential problems by taking some actions to prevent their occurrence. The purpose of self-healing is to attain system resilience and robustness by being able to recover from any failure. Self-healing implies that an autonomic system must first be capable of detecting any potential problem–for instance an unresponsive system element or a security breach. Second, it must be able to determine a viable solution for avoiding or recovering from the problem. Recovery methods include downloading software updates, restarting failed elements or simply throwing an exception to notify a human administrator. It is important to ensure that the process of self-healing does not cause further damage, such as introducing new bugs.

8.2.5 Problem Statement

A conceptual architecture of autonomic systems is proposed in [DLLF11]. Authors propose to collect all self-management components in one single entity, rather than being dispersed throughout the system software. This entity is called *autonomic manager*.

The interest of introducing the autonomic manager is clear: it allows the system to better manage the interdependence between the different components. However, it raises a serious security problem, as it simplifies the work of attackers. Indeed, attackers require only to compromise the autonomic manager in order to successfully disable both the self-protection and the self-healing mechanisms. Therefore, the security of the autonomic manager is of crucial importance. It is must be particularly protected and strongly isolated from the rest of the system, since the reliability of autonomic systems heavily depends on the proper functioning of the autonomic manager.

The problem is that the autonomic manager inherits its security from the underlying kernel. Only one exploit of this kernel would allow complete control to the entire autonomic manager. Recent incidents show that exploiting the kernel is a real threat [KPK14, XLS+15]. Hence, there is a need for security methods that provide strong protection for the software that is responsible of protecting/monitoring the system. These methods have to be properly isolated, so that they are protected even in case of any potential kernel exploitation.

Related Work

Virtualization Approaches: A large body of research, such as [ANSZ09, CLDA07, PCSL08, PH07, SLQP07], uses virtualization to provide the required isolation for securing the autonomic manager. Nevertheless, virtualization is primarily designed to allow multiple operating systems to share the same hardware platform. It is not practical, particularly in real world systems, to exclusively use the virtualization layer for the autonomic components. Furthermore, hypervisors themselves are struggling with their security problems. For instance, there were 238 reported vulnerabilities for VMware by August 2016 [CVE16b]. As a result, hosting the autonomic manager inside the hypervisor might cause fundamental security concerns.

To achieve isolation without relying on the hypervisor, research efforts are exploring two alternatives: sandboxing and hardware protection.

Sandboxing Approaches: Fides [SP12] and Inktag [HKD+13] are examples of systems that use sandboxes to isolate security sensitive code from the OS kernel. A sandbox can be used to host the autonomic manager. However, these techniques rely essentially on virtualization to provide the required isolation. Hence, they require the hypervisor to be actively involved in monitoring and managing some kernel operations. Therefore, these approaches suffer from the same fundamental problem of virtualization, which is the dependence on a flawed layer, that is the hypervisor.

Hardware Protection Approaches: Intel introduced Software Guard Extension (SGX) [CD16] that allows the execution of some verified code inside secure 'enclaves'. These enclaves are isolated from the OS kernel and isolated from each other. SGX enclaves can host the autonomic manager components without increasing the complexity of the underlying autonomic system. Nevertheless, there is no similar protection for ARM whose architecture is dominant in embedded systems [ARM15b]. Instead, ARM provides TrustZone which is, as described in section 2.4.3, a monolothic secure world that is isolated from the main kernel. Although, TZ-RKP [ANS+14], SKEE [ASB+16] and SPROBES [GJ14] proved that TrustZone can be used to monitor the kernel, these approaches are limited to only certain monitoring operations, thereby not adapted to the complex approach of the autonomic manager.

This brief review of previous work testifies that there is a problem in hosting the autonomic manager. On one hand, we argue that using the hypervisor adds risk to the hypervisor security. On the other hand, no adequate hardware-based solution is available for embedded systems.

8.2.6 Threat Model and Assumptions

Threat Model

We consider two categories of threats.

The first category concerns the OS kernel and it includes attacks that aim at executing unauthorized privileged code inside the kernel. This includes all attacks that aim to: (1) inject malicious code into the kernel, (2) modify the privileged code binaries that already exist in memory, or (3) escalate the privilege of user space code. The Self-protection mechanisms of our autonomic manager should prevent all these attacks without relying on the kernel itself.

The second category concerns the core security of the autonomic manager. It aims at compromising kernel monitor and protection mechanisms by compromising the autonomic manager itself. Our architecture guarantees that all software attacks can neither breach the isolation nor bypass the monitoring.

Assumptions

Our autonomic manager assumes an ARM-based architecture that implements the TrustZone extensions, and that it runs as part of the TEE. It also assumes that the whole system is loaded securely, including both the TEE and the Normal OS. This process is straightforward using secure boot. As a matter of fact, the secure boot is not enough for our autonomic manager, since it only guarantees the integrity of the kernel during the boot-up process. It cannot guarantee the integrity of the kernel after the system runs and starts to interact with potential attackers.

Security Guarantees

Our design provides three main security guarantees to the autonomic system.

1. It protects the autonomic manager of all software attacks by hosting it inside the TEE.
2. It self-protects the OS kernel by prohibiting from modifying the memory or access permission of the system. As a result, even if an attacker completely compromises the kernel, it would not be able to breach the access protection of the autonomic manager.
3. It guarantees that the execution of all privileged instructions passes through the autonomic manager via a strictly controlled switch gate. Thus, the autonomic manager can safely inspect input parameters passed from the kernel fro potential security threats. For instance, the autonomic manager can inspect any request to change the memory to confirm that they do not violate the guaranteed isolation.

8.2.7 Architecture

Numerous embedded systems are designed to be reliable, in the sense that they should keep on working without human intervention. With the advent of autonomic computing, reliability is achieved by self-protection and self-healing. This is true since, as previously mentioned, self-healing allows systems to recover from an internal problem, while self-protection prevents systems from being corrupted.

Here, we describe the general mechanisms upon which the autonomic manager relies in order to accomplish its tasks.

The Self-* Loop

The autonomic manager guarantees self-protection and self-healing by performing a 4-step loop.

1. *Monitoring:* to collect information about the running system by searching and reporting the used resources of the running kernel;
2. *Analysis:* to study and collect data in order to diagnose the actual state of the system;
3. *Deciding:* to adopt the strategies to be taken according to the reached conclusions of the data analysis;
4. *Execution:* to carry out the decision that were taken during the previous step.

The autonomic manager runs a permanent loop of monitoring-analysis-deciding-execution. The goal of this loop is to provide self-protection and self-healing by allowing embedded systems to protect and fix themselves. It is easy to see that each step depends strongly on the previous one to perform its task. For instance, Proper monitoring and adequate analysis are primary for the decision process that is considered as the core of the autonomic manager. Furthermore, the three first steps are based on a special component that provides three types of knowledge: (1) *state knowledge* that is used to be aware of the state of managed resources and external environment; (2) *policy knowledge* that refers to the strategies made by the human administrator or acquired through some machine learning on previous breaches; and (3) *solving knowledge* that includes various plans helping the autonomic manager take its action.

Security Mechanisms

We propose to shield the autonomic manager inside the TEE. The proposed architecture is depicted in Figure 8.2. In our architecture, the autonomic manager, which runs inside TEE, provides two security mechanisms that are triggered by two types of exceptions:

- **Software Interrupt Exceptions** that are inserted inside the kernel code and triggered before the execution of certain system instructions;

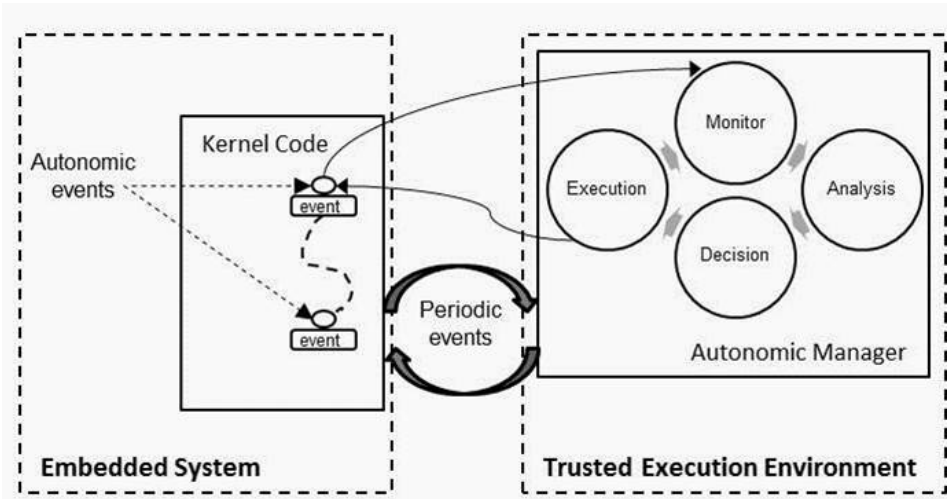


Figure 8.2: The Architecture of The Autonomic Manager

- **Periodic Exceptions** that are triggered by a secure timer that is managed by the TEE to guarantee the execution of autonomic manager after a certain amount of time.

Now, let us discuss the first security mechanism of the autonomic manager, namely **trap and emulate**. This mechanism aims at depriving the kernel from executing privileged functions, such as memory management. To this end, it forces the kernel to route through the autonomic manager for inspection and approval before executing certain instructions. This is achieved using software interrupt exceptions, that we call *autonomic events*. Self-protection is guaranteed by such security mechanism. Indeed, even a malicious kernel cannot harm the rest of the system, since the TEE must approve the execution of the critical functions. For instance, malicious attacks cannot inject code to the kernel because any write instruction to the memory must be inspected and analyzed by the autonomic manager whose protection of kernel integrity is one of its roles. A key technical challenge here is that ARM TrustZone is not capable of preventing and intercepting control instruction execution in the Normal OS. The solution is to remove certain control instructions from the kernel and replace them by *hooks* that trigger interrupt exceptions. These hooks are inserted either by code modification or binary rewriting. When the autonomic manager receives an interrupt exception, it emulates–after a classic monitoring-analysis-deciding-execution loop, the required instruction and returns the output to the kernel that continues its execution.

Regarding the second security mechanism that is **introspection**, it aims at constantly monitoring the integrity state of the loaded kernel, and thereby reacting if problems are detected. For instance, the TEE might force the system to halt running if the integrity of the kernel code is not valid. Introspection is achieved using periodic exceptions, and it allows autonomic systems to perform self-healing. We recall that self-healing is vital for embedded systems, since it provides graceful remediation to a failure in the kernel, and an unresponsive device would generate overall negative sentiments towards the system reliability. Thus, the executed introspection implements two security tools for self-healing:

1. *Watchdog function* to monitor the kernel in the Normal OS. It detects any interruption or failure of the kernel;

2. *Failure remediation* to remediate any detected failure. For instance, the most basic form of failure remediation is to restart the kernel in case it fails. Despite its simplicity, other approaches of remediation are preferred in order to avoid some interruptions of the system functions. Ensuring continuous services are of paramount importance in some sensible cases.

For introspection, relying on periodic exceptions guarantees that the mechanisms of self-healing are non-bypassable. Indeed, even if some malicious code succeeds in breaking into the system and somehow disabling the software interrupts exceptions that lead to the autonomic manager, the periodic exceptions ensure that the kernel in the Normal OS is periodically verified by the autonomic manager. These exceptions are controlled directly by the TrustZone, and bypassing them implies that the TEE was compromised. In our threat model, we suppose that the underlying TEE is immune against all software attacks. Moreover, the periodic exceptions allow the system to mitigate the threat of powerful attackers that prevent the autonomic manager to contact the system administrator. This is done by directly controlling the booting mechanism of the whole system. In the worst case, if the system cannot be recovered even by rebooting, the autonomic manager might take the decision to completely shut down the system in order to avoid further infection.

8.2.8 Open Issues

Here, we summarize some of the design challenges that we have not yet addressed in this thesis.

Control Flow Attacks

Unfortunately, the self-protection mechanisms based on trap and emulate cannot prevent all types of kernel data attacks. In particular, some type of kernel data attacks can cause the kernel control flow to change, which could lead to malicious behavior without any change in the kernel code. These attacks actually trick the kernel code into maliciously modifying its own memory. The most popular attack of such type is *return oriented attacks* [BRSS08, HHF09]. Recent research shows that these attacks are hard to prevent [GABP14, VPKE14]. The only thing that our autonomic manger can guarantee with trap and emulate that these attacks will not load new unauthorized privileged code or subvert the self-healing mechanisms. In addition, our approach can be combined with the approaches given in [CZY+14, DDE+12, ZWC+13] to completely eliminate these kind of attacks.

Concrete Security Mechanisms

In this work, we do not define the exact mechanisms for introspection or trap and emulate. Indeed, our architecture specifies the main objectives of the autonomic manager and how they can be achieved in a high-level manner. We believe that such a theoretical overview has interest in its own, as it remains applicable for any set of specific monitoring and protection mechanisms. It is important to notice that the main contribution of this work is the strong integration of the autonomic manager inside a TEE, and how ARM TrustZone extensions can be used to enrich such integration. This principle remains valid, while the technical details regarding the monitoring-analysis-deciding-execution loop of the autonomic manager change from one kernel to another.

Complete Proof-of-Concept

We admit that this work lacks of a full proof-of-concept that shows the feasibility of our architecture. This point is highly related to the previous one, since we first require a concrete set of security mechanisms protecting a specific kernel in order to provide a full-fledged proof-of-concept. Despite our arguments that we mention in the previous point, we admit the importance of having a complete implementation so as to evaluate the performance and the security of our architecture. Experimental evaluation includes (1) effectiveness against real attacks, (2) performance overhead of trap and emulate, (3) configuration settings of the periodic exceptions, (4) power consumption overhead, and (5) booting time. Embedded systems are sometimes used for critical tasks, therefore an extensive evaluation is needed. Providing the excellent performance

of ARM TrustZone in different related work [ANS+14, ASB+16, GJ14] and even in an industrial product [Seq16a], we are quite confident that a complete implementation would only confirm the relevance of our approach.

8.2.9 Conclusion

We introduced an architecture that allows autonomic systems to guarantee their principles of self-protection and self-healing. Our new solution is designed to strongly ensure protection and graceful remediation of the OS kernel. Our architecture is based on four key properties: (1) isolation from the kernel by running the autonomic manager inside a TrustZone-based TEE, (2) a limited interface to switch the context between the kernel and the autonomic manager, (3) the ability to place hooks to intercept kernel events for security inspection, and (4) periodically verify the state of the running kernel.

In addition to presenting design of our solution, we have also discussed some of the open issues related to our architecture. Our primary goal here is to ignite the discussion about new directions in trusted computing. Indeed, our work shows that TEE can be used not only to protect the execution of sensitive applications, but also to provide rich security properties (i.e. self-protection and self-healing) to a complex software component like the OS kernel.

Chapter 9

Conclusion and Perspectives

"For even the very wise cannot see all ends."

– John Ronald R. Tolkien, Lord of the Rings, The Fellowship of the Ring

This thesis focused on the security and the trust problems of smartphones. Over the last years, smartphones have reshaped every aspect of our daily life. Their transformative power comes from their mobility, ubiquity and connectivity. However, lost in the fast paced development of such devices, the security of smartphones remains a continent that is still difficult to explore due to their connected nature and the various different technologies composing them. This fact has hindered the wide adoption of smartphones in business environments requiring a high level of security. Indeed, being aware of the security problems related to their smartphones, many users still avoid relying on their devices to execute sensitive applications, such as mobile payment.

The aim of our work is to make smartphones more secure, and therefore more trustworthy to execute sensitive applications. In this thesis, we covered this *trust problem* under two complementary approaches: (1) demonstrating the limits of the used technologies by identifying security vulnerabilities in practice; and (2) enforcing the security of the sensitive applications during their execution. Below, we summarize the key learnings of our work according to their followed approach.

The relevance of provable security. We first showed the great interest of analyzing real-world systems using the paradigm of provable security. Despite being a theoretical tool, we showed that it can be handily used in order to find theoretical security flaws that could lead to serious consequences for the related system. We illustrated this point by thoroughly studying two widely deployed systems that were carefully designed to guarantee security.

The first studied system was the Android KeyStore, which is the component that shields users' cryptographic keys when stored on mobile devices. We started by formally proving that the encryption scheme used by the KeyStore does not provide integrity, which means that an adversary (an attacker in the provable security language) can undetectably modify the stored keys. Then, we pushed our investigation further and defined a concrete scenario breaching the security guaranteed by the KeyStore. In particular, the scenario allows a malicious application to make mobile apps to use weak keys, thereby making all subsequent operations involving cryptographic schemes vulnerable to exhaustive attacks.

The second system was the family of Secure Channel Protocols (SCPs). Defined by a big cross-industry consortium, SCPs play an important role in securing the content management of smart cards. In this analysis, we provided two different results. On the one hand, we demonstrated an attack against the confidentiality feature offered by SCP02, which is the most popular protocol

in the SCP family. The identified attack allows a malicious entity to recover encrypted messages during a secure session established by SCP02. On the other hand, we investigated the security of SCP03 which is, at the writing of this dissertation, the most recent SCP member that is based on symmetric keys. We found that SCP03 provably satisfies strong notions of security. More notably, it withstands replay, out-of-delivery and algorithm-substitution attacks. Thus, by taking the overall of our findings, we demonstrated how provable security can be used to both strongly guarantee the absence of security vulnerabilities and identify them if they exist. This shows the power and the versatility of such a tool.

The security offered by the Trusted Execution Environment. After applying provable security on quite simple systems, we looked at the security of more complex mobile systems and we noticed that the current approaches of enforcing mobile security are not enough to offer a tamper-resistance environment for sensitive applications. However, one of the identified approaches seemed promising. It consists of splitting the mobile system into two strongly isolated subsystems, one of which is trustworthy and the other one runs the full-fledged mobile operating system. Not yet well formalized, we investigated this approach and showed its untapped potential in terms of security.

We first revisited the aforementioned approach and called it the Dual-Execution-Environment (Dual-EE) approach. To this end, we proposed a theoretical framework to systematize the design of dual-EE solutions regarding well-established primitives defined in the architecture of the Multiple Independent Levels of Security (MILS). This framework allowed us to soundly evaluate the existing dual-EE technologies, thereby helping us identifying the best technology to use for our subsequent work. That technology was ARM TrustZone that we combined together with the dual-EE approach in order to build a *trusted execution environment*.

Then, we kept our investigation by studying the technology coined by GlobalPlatform as the Trusted Execution Environment (TEE). TEE is commonly known as an isolated processing environment in which applications can be securely executed irrespective of the rest of the system. Despite its wide deployment, existing definitions of TEE are largely inconsistent and unspecific, which leads to confusion in the use of the term and its differentiation from related concepts. Therefore, and in order to better understand the extent of this technology, we proceeded by proposing a precise definition of TEE. Furthermore, we extensively analyzed its core properties by identifying the most relevant approaches to achieve them. This abstract model allowed us to delve into the existing TEE solutions to identify their strength and their weakness.

One of the recurrent weaknesses in TEE solutions is the security level guaranteed by the TEE. Indeed, several attacks have been recently disclosed against deployed TEE systems, which jeopardize the security promise of such a technology. When compared to the tamper-resistant Secure Element, TEE is often perceived as a tradeoff between more computation power, but less security. Thus, we ended up our thesis work by exploring a new direction in trusted computing that allows TEE to protect its sensitive data even when the secure kernel gets compromised. We achieved this by using efficient techniques for two-party secure function evaluation. The main interest of our architecture is to weaken the *trust requirement* for the TEE. Thus, Service Providers, like Operators or Banks, can benefit from the TEE security and power without having to fully trust the TEE issuer. Such an execution model for the TEE could raise the TEE adoption, since it improves security and simplifies the installation process of secure applications.

Perspectives.

Our work in this thesis has shown that even a secure protocol that was highly certified by an international institution still contains some security flaws. The fact that these flaws are caused by well-known cryptographic vulnerabilities (the use of a constant IV for CBC mode) raises a serious question about the trustworthiness of such a protocol. Indeed, SCP02 uses Triple DES

(3DES) with only two independent keys. Mitchell in [Mit16] shows that the security margin of 2-key 3DES is slim by giving further enhancements to some attacks undermining the 2-key 3DES. Moreover, SCP02 uses ISO9797-1 MAC algorithm 3 whose security has been questioned in [FT14]. We believe that it would be interesting to combine our attack with these recent works in order to demonstrate a feasible attack against smart cards that can be conducted in practice. Such an attack would have severe consequences, since it would concern many deployed smart cards, and more importantly smart card issuers cannot dynamically update the SCP implementations.

As regards the TEE architectures, we have presented a set of open issues related to our work of integrating Garbled Circuits (GC) into the TEE design. Of particular importance, Authors in [MGC+16] show that the existing GC frameworks are unreliable by demonstrating their unpredictable behavior in many cases. Therefore, we emphasize that there is a need to GC frameworks whose correctness was validated through principled design. One area for future research is to apply formal verification tools to produce a formally *verified implementation* of the primary components of our architecture. Moreover, Our use of component-based GC offers significant benefits in terms of efficiency and computation time, but it comes at a noticeable cost to storage size. This means that the size of our architecture might become prohibitive for mobile devices in case many basic functions are needed. Finding the correct optimization for the storage size of GC is a challenging issue.

Finally, we have shown the interest that the TEE presents for smartphones in this thesis. It would be interesting to apply the architectures introduced in our work in the context of other embedded devices. In particular, the design of modern smart vehicles includes the same security problems as for smartphones. Indeed, a smart vehicle executes two systems that do not share the same level of security: one runs entertainment services, like music and Internet connection, and the other controls driving, like the brake system. Due to its connected and open nature, the entertainment system is much likely to be compromised than the driving system which is more closed. Therefore, the later should be strongly isolated from the former in order to avoid security breaches that could result in car accidents. Moreover, modern set-up boxes (STB) also present an interesting area to apply the design principles of TEE. A modern STB might contain applications installed on-the-fly by their users to enrich their TV experience. At the same time, an STB still hosts secrets that help their issuers performing some sensitive applications. Therefore, a new generation of TEE-based STBs may open the door toward more customization and more services offered to clients. The scope to these services might include securing IoT (Internet of Things) communications inside the *Smart Home*.

List of Publications

The list of our publications during this thesis is given below. This dissertation discusses the content of all of them except for the paper *Over-the-Internet: Efficient Remote Content Management for Secure Elements in Mobile Devices* [SAB15b], which was presented at the conference MobiSecServ 2015. In this paper, we have considered the problem of managing applications in the NFC ecosystem. Indeed, too often, service providers still rely on slow and unreliable technologies, such as SMS-based Over-the-Air (OTA), to install and update their NFC applications. In order to solve this problem, we have proposed OTI (Over-the-Internet) that handles the management of NFC applications in a secure and fast way. In addition, we have designed OTI to be compliant with the existing NFC standards. OTI improves the efficiency of installing new applications by 90%, which is considered as a significant gain. For instance, OTI requires 25 seconds only to install an application of 9 kilobytes, while SMS-based OTA requires 5 minutes to install the same application. Despite its great interest, the subject of this paper was not included here, since it does not fit well within the topics discussed in this dissertation.

Invited Conferences

[Sab16] *Trusted Execution Environment: Trusted Model, Architecture Overview, Challenges and Future Directions* (**MobiSecServ 2016**)
Mohamed Sabt.

International Conferences with Review and Proceeding

[ST16b] *Cryptanalysis of GlobalPlatform Secure Channel Protocols* (**SSR 2016**)
Mohamed Sabt and Jacques Traoré.

[ST16a] *Breaking Into the KeyStore: A Practical Forgery Attack Against Android KeyStore* (**ESORICS 2016**)
Mohamed Sabt and Jacques Traoré.

[SAB15d] *Trusted Execution Environment: What It Is, and What It Is Not* (**TRUSTCOM 2015**)
Mohamed Sabt, Mohammed Achemlal and Abdelmadjid Bouabdallah.

[SAB15a] *The Dual-Execution-Environment Approach: Analysis and Comparative Evaluation* (**IFIP SEC 2015**)
Mohamed Sabt, Mohammed Achemlal and Abdelmadjid Bouabdallah.

[SAB15b] *Over-the-Internet: Efficient Remote Content Management for Secure Elements in Mobile Devices* (**MobiSecServ 2015**)
Mohamed Sabt, Mohammed Achemlal and Abdelmadjid Bouabdallah.

Posters

[SAB15c] *Towards Integrating Trusted Execution Environment into Embedded Autonomic Systems* (**ICAC 2015**)
Mohamed Sabt, Mohammed Achemlal and Abdelmadjid Bouabdallah.

Ongoing Work

[CDK+16] *BlindIDS: Market-Compliant, Privacy-Friendly and Security-Aware Intrusion Detection Systems over Encrypted Traffic*
Sébastien Canard, Aïda Diop, Nizar Kheir, Marie Paindavoine and Mohamed Sabt.

[SA16a] *Garbling the Trust: Trusted Execution Environment Based on Garbled Circuits*
Mohamed Sabt and Mohammed Achemlal.

[SA16c] *uStore: Resource-Efficient UICC-Based Android KeyStore in Practice*
Mohamed Sabt and Mohammed Achemlal.

Patents

The findings of this thesis have given two internationally published patents. Both of them concern the NFC technology. The first one is about the Host Card Emulation (HCE) technology, and the second is about the relay attack. Despite their interest, the two patents were not included here, since they do not fit well within the topics discussed in this disseration.

[SAA16] *Procédé de sécurisation de transactions sans contact* (**WO/2016/102831**)
Mohamed Sabt, Mouhannad Alattar and Mohammed Achemlal.

[SA16b] *Procédé de protection d'un terminal mobile contre des attaques (**WO/2016/051059**)*
Mohamed Sabt and Mohammed Achemlal.

List of Tables

6.1 Summary of Our Comparative Evaluation of Dual-EE Solutions 111

7.1 A Comparison Study of the Six Representative TEE Solutions 131

List of Figures

2.1 Android System Architecture [And16a] 20
2.2 Binder Communication 21
2.3 Android KeyStore Architecture 23
2.4 Overview of the APDU Structure 27
2.5 ARMv8 Execution Levels [ARM15a] 28
2.6 An Overview of the ARM TrustZone Technology 29

3.1 Overview of Proofs by Reduction 42

4.1 Overview of the Forgery Attack Against the Android KeyStore 64

5.1 Ciphertext Generation by SCP02. Grey boxes, i.e. 'MAC padding', are not included in the 'encrypt' operation. 75
5.2 Ciphertext Generation by SCP03. 80

6.1 An Overview of the MILS Architecture 103
6.2 Representation of the Dual-EE Approach in the MILS Abstraction 106
6.3 Proof by Picture When Multi-EE Runs Four Execution Environments 106

7.1 An Overview of TEE Building Blocks 122

8.1 TrustedGarble Architecture 142
8.2 The Architecture of The Autonomic Manager 152

Acronyms

2FA Two-Factor Authentication.

3DES Triple DES.

3GPP 3rd Generation Partnership Project.

AE Authenticated Encryption.

AEAD Authenticated Encryption with Associated Data.

AES Advanced Encryption Standard.

APDU Application Protocol Data Unit.

API Application Programming Interface.

ASA Algorithm Substitution Attack.

AT Hayes Attention.

CBC Cipher Block Chaining.

CC Common Criteria.

CCA Chosen-Ciphertext Attack.

CPA Chosen-Plaintext Attack.

CPU Central Processing Unit.

CTR Counter.

CUF-CPA Ciphertext Unforgeability.

DES Data Encryption Standard.

DMA Direct Memory Access.

DRTM Dynamic Root of Trust Measurement.

DSA Digital Signature Algorithm.

EAL Evaluation Assurance Level.

EaM Encrypt-and-MAC.

EAP Extensible Authentication Protocol.

EC Elliptic Curve.

ECB Electronic Code Book.

eMMC embedded MultiMedia Card.

EMV Europay, MasterCard, and Visa.

EtM Encrypt-then-MAC.

EwR Encryption-with-Redundancy.

FIFO First In, First Out.

G&D Giesecke & Devrient GmbH.

GC Garbled Circuit.

GCM Galois Counter Mode.

GID Group ID.

GP GlobalPlatform.

GPEE General-Purpose Execution Environment.

GSM Global System for Mobile Communications.

HMAC Hash based MAC.

HtCBC Hash-then-CBC-Encrypt.

HtCTR Hash-then-CTR-Encrypt.

HtE Hash-then-Encrypt.

HW Hardware.

I/O Input/Output.

IND Indistinguishability.

INT-CTXT Integrity of Ciphertext.

IoT Internet of Things.

IPC Inter-Process Communication.

ISP Issuer Security Domain.

IV Initialization Vector.

JCA Java Cryptography Architecture.

JCVM JavaCard Virtual Machine.

JNI Java Native Interface.

JTAG Joint Test Action Group.

JVM Java Virtual Machine.

LOC Lines of code.

LSB Least Significant Bit.

LTE Long-Term Evolution.

MAC Message Authentication Code.

MD5 Message-Digest Algorithm 5.

MILS Multiple Independent Levels of Security.

MMS Multimedia Messaging Service.

MSB Most Significant Bit.

MtE MAC-then-Encrypt.

NDA Non-Disclosure Agreement.

NFC Near Field Communication.

NIST National Institute of Standards and Technology.

ObC On-board Credentials.

OMAC One-Key CBC-MAC.

OMTP Open Mobile Terminal Platform.

OP-TEE Open Portable TEE.

OS Operating System.

OT Oblivious Transfer.

OTP One Time Password.

PIN Personal Identification Number.

PRF Pseudorandom Function Family.

PRP Pseudorandom Permutation Family.

PUF Physically Unclonable Function.

QSEE Qualcomm Secure Execution Environment.

RAM Random Access Memory.

ROM Read-Only Memory.

RoT Root of Trust.

RPC Remote Procedure Call.

RPMB Replay-Protected Memory Blocks.

RSA Rivest-Shamir-Adleman.

RTOS Real-Time Operating System.

SCP Secure Channel Protocols.

SCR Secure Configuration Register.

SD Security Domain.

SE Secure Element.

SEE Secure Execution Environment.

SFE Secure Function Evaluation.

SHA Secure Hash Algorithm.

SIM Subscriber Identity Module.

SK Separation Kernel.

SKPP Separation Kernel Protection Profile.

SMC Software Monitor Call.

SMS Short Message Service.

SSH Secure Shell.

SUF Strong Unforgeability.

TaGa Try-and-Guess Attack.

TBC Tweakable Block Cipher.

TCB Trusted Computing Base.

TEE Trusted Execution Environment.

TLK Trusted Little Kernel.

TLR Trusted Language Runtime.

TPM Trusted Platform Module.

TSM Trusted Service Manager.

TUI Trusted User Interface.

UI User Interface.

UICC Universal Integrated Circuit Card.

UID Unique ID.

VM Virtual Machine.

VPN Virtual Private Network.

Bibliography

[3GP16] 3GPP. The mobile broadband standard, 2016. `http://www.3gpp.org/`.

[AB01] Jee Hea An and Mihir Bellare. Does encryption with redundancy provide authenticity? In *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 512–528. Springer-Verlag, 2001.

[ABF+03] Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter, and Jean-Pierre Seifert. Fault attacks on rsa with crt: Concrete results and practical countermeasures. In *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 260–275. Springer-Verlag, 2003.

[ABN+12] Nicolas Anciaux, Luc Bouganim, Benjamin Nquyen, Iulian Sandu Popa, Philippe Pucheral, and Philippe Bonnet. *Trusted Cells: A Sea Change for Personal Data Services*. ITU Technical Report Series. IT-Universitetet i København, 2012.

[AEK+14] N. Asokan, Jan-Erik Ekberg, Kari Kostiainen, Anand Rajan, Carlos V. Rozas, Ahmad-Reza Sadeghi, Steffen Schulz, and Christian Wachsmann. Mobile trusted computing. *Proceedings of the IEEE*, 102(8):1189–1206, August 2014.

[AF12] João Alfaiate and José Fonseca. Bluetooth security analysis for mobile phones. In *Proceedings of the 7th Iberian Conference on Information Systems and Technologies*, CISTI '12, pages 1–6. IEEE, 2012.

[AFA+13] Zaheer Ahmad, Lishoy Francis, Tansir Ahmed, Christopher Lobodzinski, Dev Audsin, and Peng Jiang. Enhancing the security of mobile applications by using tee and (u)sim. In *Proceedings of the 10th IEEE International Conference on Autonomic and Trusted Computing*, UIC/ATC '13, pages 575–582. IEEE, 2013.

[AFHOT06] Jim Alves-Foss, W. Scott Harrison, Paul Oman, and Carol Taylor. The mils architecture for high-assurance embedded systems. *International Journal of Embedded Systems*, 2:239–247, 2006.

[AFS97] William A. Arbaugh, David J. Farber, and Jonathan M. Smith. A secure and reliable bootstrap architecture. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, SP '97, pages 65–71. IEEE, 1997.

[AGS83] Stanley R. Jr. Ames, Morrie Gasser, and Roger R. Schell. Security kernel design and implementation: An introduction. *Computer*, 16(7):14–22, July 1983.

[AGT14] Ghada Arfaoui, Saïd Gharout, and Jacques Traoré. Trusted execution environments: A look under the hood. In *Proceedings of the 2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, MobileCloud '14, pages 259–266. IEEE, 2014.

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 535–548. ACM, 2013.

[ALZR13] Waqar Anwar, Dale Lindskog, Pavol Zavarsky, and Ron Ruhl. Redesigning secure element access control for nfc enabled android smartphones using mobile trusted computing. In *Proceedings of the 2013 International Conference on Information Society*, i-Society '13, pages 27–34. IEEE, 2013.

[AM16] Raja Naeem Akram and Konstantinos Markantonakis. *Challenges of Security and Trust of Mobile Devices as Digital Avionics Component*, pages 1C4–1–1C4–11. ICNS '16. IEEE, 2016.

[And16a] Android. Android interfaces and architecture, 2016. `https://source.android.com/devices/`.

[And16b] Android. Android open source project – issue tracker, January 2016. `https://code.google.com/p/android/issues/entry`.

[And16c] Android. Application security – sim card access, 2016. `https://source.android.com/security/overview/app-security.html#sim-card-access`.

[And16d] Android. Keystore implementation, 2016. `https://android.googlesource.com/platform/system/security/+/master/keystore/keystore.cpp`.

[And16e] Android. Trusty tee, 2016. `https://source.android.com/security/trusty/`.

[ANS+14] Ahmed M. Azab, Peng Ning, Jitesh Shah, Quan Chen, Rohan Bhutkar, Guruprasad Ganesh, Jia Ma, and Wenbo Shen. Hypervision across worlds: Real-time kernel protection from the arm trustzone secure world. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 90–102. ACM, 2014.

[ANSZ09] Ahmed M. Azab, Peng Ning, Emre C. Sezer, and Xiaolan Zhang. Hima: A hypervisor-based integrity measurement agent. In *Proceedings of the 2009 IEEE Annual Conference on Computer Security Applications*, CCS '09, pages 461–470. IEEE, 2009.

[AP12] Matthew Areno and Jim Plusquellic. Securing trusted execution environments with puf generated secret keys. In *Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, TRUSTCOM '12, pages 1188–1193. IEEE, 2012.

[ARM09] ARM Security Technology. Building a secure system using trustzone technology. Technical Report PRD29-GENC-009492C, ARM, 2009.

[ARM15a] ARM. Arm cortex-a series – programmer's guide for armv8-a, March 2015. `https://static.docs.arm.com/den0024/a/DEN0024.pdf`.

[ARM15b] ARM Holdings plc. Annual report 2015: Strategic report, 2015. `http://ir.arm.com/phoenix.zhtml?c=197211&p=irol-reportsannual`.

[ARM16a] ARM. Company profile, 2016. `http://www.arm.com/about/company-profile/`.

[Arm16b] Lucian Armasu. All the new security improvements in android n, May 2016. `http://www.tomshardware.co.uk/google-android-n-security-improvements,news-53030.html`.

[ASB+16] Ahmed M. Azab, Kirk Swidowski, Rohan Bhutkar, Jia Ma, Wenbo Shen, Ruowen Wang, and Peng Ning. Skee: A lightweight secure kernel-level execution environment for arm. In *Proceedings of the 23rd Annual Network and Distributed System Security Symposium*, NDSS '16. Internet Society, 2016.

[BØ5] Santiago Zanella Béguelin. Formalisation and verification of the globalplatform card specification using the b method. In *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices*, volume 3956 of *Lecture Notes in Computer Science*, pages 155–173. Springer-Verlag, 2005.

[Bar06] Gregory V. Bard. A challenging but feasible blockwise-adaptive chosen-plaintext attack on ssl. In *Proc. of the International Conference on Security and Cryptography*, SECRYPT '06, pages 7–10. INSTICC Press, 2006.

[BDJR97] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, FOCS '97, pages 394–403. IEEE, 1997.

[Bel99] Mihir Bellare. Practice-oriented provable-security. In *Lectures on Data Security*, volume 1561 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 1999.

[Ber15] Dan J. Bernstein. CAESAR: Competition for authenticated encryption: Security, applicability, and robustness, December 2015. `http://competitions.cr.yp.to/caesar.html`.

[BFC09] Reinhardt A. Botha, Steven M. Furnell, and Nathan L. Clarke. From desktop to mobile: Examining the security experience. *Computer Security*, 28(3-4):130–137, May 2009.

[BFH+11] Michael Becher, Felix C. Freiling, Johannes Hoffmann, Thorsten Holz, Sebastian Uellenbeck, and Christopher Wolf. Mobile security catching up? revealing the nuts and bolts of the security of mobile devices. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, pages 96–111. IEEE, 2011.

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 478–492. IEEE, 2013.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 784–796. ACM, 2012.

[BJK15] Mihir Bellare, Joseph Jaeger, and Daniel Kane. Mass-surveillance without the state: Strongly undetectable algorithm-substitution attacks. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1431–1440. ACM, 2015.

[BKN02] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Authenticated encryption in ssh: Provably fixing the ssh binary packet protocol. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 1–11. ACM, 2002.

[BKR00] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, December 2000.

[BMR90] Donald Beaver, Silvio Micali, and Philippe Rogaway. The round complexity of secure protocols. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 503–513. ACM, 1990.

[BN08] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of Cryptology*, 21(4):469–491, September 2008.

[BPR14] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In *Advances in Cryptology – CRYPTO 2014*, volume 8616 of *Lecture Notes in Computer Science*, pages 1–19. Springer-Verlag, 2014.

[Bra16] Pascal Brand. Op-tee, 2016. `https://github.com/OP-TEE/`.

[Bro16] Robert Brodie. The problems weighing down mobile payments, April 2016. `http://www.mobilepaymentstoday.com/articles/the-problems-weighing-down-mobile-payments/`.

[BRSS08] Erik Buchanan, Ryan Roemer, Hovav Shacham, and Stefan Savage. When good instructions go bad: Generalizing return-oriented programming to risc. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, pages 27–38. ACM, 2008.

[BRW04] Mihir Bellare, Phillip Rogaway, and David Wagner. The eax mode of operation. In *Fast Software Encryption – FSE 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 389–407. Springer-Verlag, 2004.

[BU02] John Black and Hector Urtubia. Side-channel attacks on symmetric encryption schemes: The case for authenticated encryption. In *Proceedings of the 11th USENIX Security Symposium*, SSYM '02, pages 327–338. USENIX Association, 2002.

[CD16] Victor Costan and Srinivas Devadas. Intel sgx explained. Cryptology ePrint Archive, Report 2016/086, 2016.

[CDA14] John Criswell, Nathan Dautenhahn, and Vikram Adve. Virtual ghost: Protecting applications from hostile operating systems. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 81–96. ACM, 2014.

[CDK+16] Sébastien Canard, Aïda Diop, Nizar Kheir, Marie Paindavoine, and Mohamed Sabt. Blindids: Market-compliant, privacy-friendly and security-aware intrusion detection systems over encrypted traffic. unpublished, 2016.

[CdRP14] Tim Cooijmans, Joeri de Ruiter, and Erik Poll. Analysis of secure key storage solutions on android. In *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, SPSM '14, pages 11–20. ACM, 2014.

[Che00] Zhiqun Chen. *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[CL10] David Champagne and Ruby B. Lee. Scalable architectural support for trusted software. In *Proceedings of the Sixteenth International Symposium on High-Performance Computer Architecture*, HPCA '10, pages 1–12. IEEE, 2010.

[CLDA07] John Criswell, Andrew Lenharth, Dinakar Dhurjati, and Vikram Adve. Secure virtual architecture: A safe execution environment for commodity operating systems. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 351–366. ACM, 2007.

[Com16a] Federal Communications Commission. Equipment authorization procedures, 2016. `https://www.fcc.gov/general/equipment-authorization-procedures`.

[Com16b] Common Criteria. Certified products, 2016. `https://www.commoncriteriaportal.org/products/`.

[COO11] Vedat Coskun, Kerem Ok, and Busra Ozdenizci. *Near Field Communication (NFC): From Theory to Practice*. Wiley, 2011.

[CPST15] Sébastien Canard, David Pointcheval, Olivier Sanders, and Jacques Traoré. Divisible e-cash made practical. In *Public-Key Cryptography – PKC 2015*, volume 9020 of *Lecture Notes in Computer Science*, pages 77–100. Springer-Verlag, 2015.

[Cro01] Paul Crowley. Mercy: A fast large block cipher for disk sector encryption. In *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 49–63. Springer-Verlag, 2001.

[CRSP11] Siddhartha Chhabra, Brian Rogers, Yan Solihin, and Milos Prvulovic. Secureme: A hardware-software approach to full system security. In *Proceedings of the International Conference on Supercomputing*, ICS '11, pages 108–119. ACM, 2011.

[CVE16a] CVE Details. Android: Security vulnerabilities, September 2016. `http://www.cvedetails.com/vulnerability-list/vendor_id-1224/product_id-19997/Google-Android.html`.

[CVE16b] CVE Details. Vmware: Vulnerability statistics, August 2016. `http://www.cvedetails.com/vendor/252/Vmware.html`.

[CXS+05] Shuo Chen, Jun Xu, Emre C. Sezer, Prachi Gauriar, and Ravishankar K. Iyer. Non-control-data attacks are realistic threats. In *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, SSYM '05, pages 12–12. USENIX Association, 2005.

[CZ11] Istehad Chowdhury and Mohammad Zulkernine. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. *Journal of Systems Architecture*, 57(3):294–313, March 2011.

[CZY+14] Yueqiang Cheng, Zongwei Zhou, Miao Yu, Xuhua Ding, and Robert H. Deng. Ropecker: A generic and practical approach for defending against rop attacks. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium*, NDSS '14. Internet Society, 2014.

[Dai02] Wei Dai. An attack against ssh2 protocol, February 2002. Email to the SECSH Working Group available from `ftp://ftp.ietf.org/ietf-mail-archive/secsh/2002-02.mail`.

[DDE+12] Lucas Davi, Alexandra Dmitrienko, Manuel Egele, Thomas Fischer, Thorsten Holz, Ralf Hund, Stefan Nürnberger, and Ahmad-Reza Sadeghi. Mocfi: A framework to mitigate control-flow attacks on smartphones. In *Proceedings of the 19th Annual Network and Distributed System Security Symposium*, NDSS '12. Internet Society, 2012.

[Den08] Alexander W. Dent. A brief history of provably-secure public-key encryption. In *Advances in Cryptology – AFRICACRYPT 2008*, volume 5023 of *Lecture Notes in Computer Science*, pages 357–370. Springer-Verlag, 2008.

[Dep85] Department of Defense Standard. Trusted computer system evaluation criteria. Technical Report DoD 5200.28-STD, U.S. Department of Defense, December 1985.

[DH79] Whitfield Diffie and Martin E. Hellman. Privacy and authentication: An introduction to cryptography. *Proceedings of the IEEE*, 67(3):397–427, 1979.

[DLC+12] Jiun-hung Ding, Chang-jung Lin, Ping-hao Chang, Chieh-hao Tsang, Wei-chung Hsu, and Yeh-ching Chung. Armvisor: System virtualization for arm. In *Proceedings of the Ottawa Linux Symposium*, OLS '12, pages 93–107, 2012.

[DLLF11] Fu Duan, Xiaoli Li, Yi Liu, and Yun Fang. Towards autonomic computing: A new self-management method. In *Artificial Intelligence and Computational Intelligence – AICI 2011 – Part 1*, volume 7002 of *Lecture Notes in Computer Science*, pages 292–299. Springer-Verlag, 2011.

[DLM+14] Joshua J. Drake, Zach Lanier, Collin Mulliner, Pau Oliva Fora, Stephen A. Ridley, and Georg Wicherski. *Android Hacker's Handbook*. Wiley Publishing, 1st edition, 2014.

[DN14] Christoffer Dall and Jason Nieh. Kvm/arm: The design and implementation of the linux arm hypervisor. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 333–348. ACM, 2014.

[Dom11] Karl Dominguez. Trojanized apps root android devices, March 2011. `http://www.zdnet.com/article/researchers-expose-android-webkit-browser-exploit/`.

[DPW11] Jean Paul Degabriele, Kenny Paterson, and Gaven Watson. Provable security in the real world. *IEEE Security and Privacy*, 9(3):33–41, May 2011.

[DR11] Thai Duong and Juliano Rizzo. Here come the xor ninjas. unpublished, 2011.

[DSZ14] Daniel Demmler, Thomas Schneider, and Michael Zohner. Ad-hoc secure two-party computation on mobile devices using hardware tokens. In *Proceedings of the 23rd USENIX Security Symposium*, SSYM '14, pages 893–908. USENIX Association, 2014.

[DTH06] Rachna Dhamija, J. D. Tygar, and Marti Hearst. Why phishing works. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pages 581–590. ACM, 2006.

[DW09] Kurt Dietrich and Johannes Winter. Implementation aspects of mobile and embedded trusted computing. In *Trusted Computing – TRUST 2009*, volume 5471 of *Lecture Notes in Computer Science*, pages 29–44. Springer-Verlag, 2009.

[DW10] Kurt Dietrich and Johannes Winter. Towards customizable, application specific mobile trusted modules. In *Proceedings of the Fifth ACM Workshop on Scalable Trusted Computing*, STC '10, pages 31–40. ACM, 2010.

[Dwo01a] Morris Dworkin. Recommendation for block cipher modes of operation: Methods and techniques. National Institute of Standards and Technology (NIST), December 2001. NIST Special Publication 800-38A.

[Dwo01b] Morris Dworkin. Recommendation for block cipher modes of operation: The cmac mode for authentication. National Institute of Standards and Technology (NIST), November 2001. NIST Special Publication 800-38B.

[EH13] Kevin Elphinstone and Gernot Heiser. From l3 to sel4 what have we learnt in 20 years of l4 microkernels? In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 133–150. ACM, 2013.

[EHKM11] David Evans, Yan Huang, Jonathan Katz, and Lior Malka. Efficient privacy-preserving biometric identification. In *Proceedings of the 18th Network and Distributed Security Symposium*, NDSS '11. Internet Society, 2011.

[Ele15] Nikolay Elenkov. Hardware-accelerated disk encryption in android 5.1, May 2015. `https://nelenkov.blogspot.fr/2015/05/hardware-accelerated-disk-encryption-in.html`.

[EMV] EMVCo. EMVCo Specification. `https://www.emvco.com/specifications.aspx`.

[EMV07] EMVCo. Emv card personalization specification – version 1.1, July 2007. `https://www.emvco.com/specifications.aspx?id=20`.

[EOM09] William Enck, Machigar Ongtang, and Patrick McDaniel. Understanding android security. *IEEE Security Privacy*, 7(1):50–57, January 2009.

[EST14] Mohamed Ali El-Serngawy and Chamseddine Talhi. Securing business data on android smartphones. In *Mobile Web Information Systems – MobiWIS 2014*, volume 8640 of *Lecture Notes in Computer Science*, pages 218–232. Springer-Verlag, 2014.

[ETS11] ETSI 3GPP. Digital cellular telecommunications system (phase 2+) – universal mobile telecommunications system (umts) – lte – at command set for user equipment (ue). Technical report, European Telecommunications Standards Institute, April 2011.

[FAWH15] Andreas Fitzek, Florian Achleitner, Johannes Winter, and Daniel Hein. The andix research os – arm trustzone meets industrial control systems security. In *Proceedings of the 2015 IEEE 13th International Conference on Industrial Informatics*, INDIN '15, pages 88–93. IEEE, 2015.

[FBL+15] Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan. Android security: A survey of issues, malware penetration, and defenses. *IEEE Communications Surveys Tutorials*, 17(2):998–1022, Second quarter 2015.

[Fei73] Horst Feistel. *Cryptography and Computer Privacy.* Scientific American, 1973.

[FFC+11] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, pages 3–14. ACM, 2011.

[Fin05] Financial. Secure payment via mobiles. *Card Technology Today*, 17(3):5–5, March 2005.

[Fis15] Kathleen Fisher. Using formal methods to eliminate exploitable bugs. In *Proceedings of the 24th USENIX Conference on Security*, SEC '15, pages 24–24. USENIX Association, 2015.

[FJMV03] Pierre-Alain Fouque, Antoine Joux, Gwenaelle Martinet, and Frederic Valette. Authenticated on-line encryption. In *Selected Areas in Cryptography – SAC 2003*, volume 3006 of *Lecture Notes in Computer Science*, pages 145–159. Springer-Verlag, 2003.

[FM06] Stefan Frei and Martin May. The speed of (in)security. In *Blackhat USA 2006*. BlackHat, 2006.

[FP98] Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co., 3rd edition, 1998.

[FT14] Benoit Feix and Hugues Thiebeauld. Defeating iso9797-1 mac algo 3 by combining side-channel and brute force techniques. Cryptology ePrint Archive, Report 2014/702, 2014.

[GABP14] Enes Göktas, Elias Athanasopoulos, Herbert Bos, and Georgios Portokalidis. Out of control: Overcoming control-flow integrity. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 575–589. IEEE, 2014.

[Gas88] Morrie Gasser. *Building a Secure Computer System*. Van Nostrand Reinhold Co., 1988.

[GEN14] GENODE. An exploration of arm trustzone technology, 2014. `https://genode.org/documentation/articles/trustzone`.

[GHL+07] David Goldenberg, Susan Hohenberger, Moses Liskov, Elizabeth Crump Schwartz, and Hakan Seyalioglu. On tweaking luby-rackoff blockciphers. In *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 342–356. Springer-Verlag, 2007.

[Gie08] Giesecke & Devrient GmbH. Mobicore4 (mc4) runtime for arm trustzone technology, August 2008. `http://www.funkschau.de/fileadmin/media/whitepaper/files/Mobicore_Whitepaper_final_.pdf`.

[Gie15] Giesecke & Devrient GmbH. Android mobicore driver for exynos 5, August 2015. `https://android.googlesource.com/platform/hardware/samsung_slsi/exynos5/+/master/mobicore/common/MobiCore/inc/mcSo.h`.

[Gie16] Giesecke & Devrient GmbH. Seek for android – secure element evaluation kit for the android platform, 2016. `http://seek-for-android.github.io/`.

[GJ14] Xinyang Ge and Trent Jaeger. Sprobes: Enforcing kernel code integrity on the trustzone architecture. In *Proceedings of the 2014 Workshop on Mobile Security Technologies*, MoST '14. IEEE, 2014.

[GKP+13] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 555–564. ACM, 2013.

[GLMY16] Adam Groce, Alex Ledger, Alex J. Malozemoff, and Arkady Yerukhimovich. Compgc: Efficient offline/online semi-honest two-party computation. Cryptology ePrint Archive, Report 2016/458, 2016. `http://eprint.iacr.org/2016/458`.

[Glo10] GlobalPlatform. Tee client api specification v 1.0, July 2010. `http://www.globalplatform.org/specificationsdevice.asp`.

[Glo11a] GlobalPlatform. Tee internal api specification v 1.0, December 2011. http://www.globalplatform.org/specificationsdevice.asp.

[Glo11b] GlobalPlatform. Tee system architecture, December 2011. www.globalplatform.org/specificationsdevice.asp.

[Glo13] GlobalPlatform. Trusted user interface api v 1.0, June 2013. http://www.globalplatform.org/specificationsdevice.asp.

[Glo14] GlobalPlatform. Secure channel protocol '3' – card specification v 2.2 – amendment d v 1.1.1, July 2014. http://www.globalplatform.org/specificationscard.asp.

[Glo15a] GlobalPlatform. Globalplatform card specification v 2.3, October 2015. http://www.globalplatform.org/specificationscard.asp.

[Glo15b] GlobalPlatform Device Committee. Tee protection profile – version 1.2. Technical Report GPD_SPE_021, GlobalPlatform, 2015.

[Glo16a] GlobalPlatform. About globalplatform – security task force activities and achievements – 2016 activities and priorities, 2016. https://www.globalplatform.org/aboutustaskforcesSecurity.asp.

[Glo16b] GlobalPlatform. Globalplatform made simple guide: Trusted execution environment (tee) guide, 2016. http://www.globalplatform.org/mediaguidetee.asp.

[Glo16c] GlobalPlatform. The standard for managing applications on secure chip technology, 2016. https://www.globalplatform.org.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[GM14] Johannes Götzfried and Tilo Müller. Analysing android's full disk encryption feature. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, 5(1):84–100, March 2014.

[Gmb14] NXP Semiconductors Germany Gmbh. Nxp j3e081_m64, j3e081_m66, j2e081_m64, j3e041_m66, j3e016_m66, j3e016_m64, j3e041_m64 secure smart card controller. Common Criteria for Information Technology Security Evaluation, August 2014. Certification Report: NSCIB-CC-13-37761-CR2.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.

[GPC+03] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 193–206. ACM, 2003.

[GPHB11] Kevin Gudeth, Matthew Pirretti, Katrin Hoeper, and Ron Buskey. Delivering secure applications on commercial mobile devices: The case for bare metal hypervisors. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, pages 33–38. ACM, 2011.

[Gra06] David Grawrock. *The Intel Safer Computing Initiative: Building Blocks for Trusted Computing*. Books by engineers, for engineers. Intel Press, 2006.

[Gro14] Trusted Computing Group. Trusted platform module library – part 1: Architecture – family "2.0", October 2014. http://www.trustedcomputinggroup.org/tpm-library-specification/.

[GRS13] Nico Golde, Kévin Redon, and Jean-Pierre Seifert. Let me answer that for you: Exploiting broadcast information in cellular networks. In *Proceedings of the 22nd USENIX Conference on Security*, SEC '13, pages 33–48. USENIX Association, 2013.

[GWV03] David Greve, Matthew Wilding, and W. Mark Vanfleet. A separation kernel formal security policy. In *Proceedings of the Fourth International Workshop on the ACL2 Theorem Prover and its Applications*, ACL2 '03, 2003.

[Hat97] Les Hatton. Reexamining the fault density-component size connection. *IEEE Software*, 14(2):89–97, March 1997.

[HD14] Roee Hay and Avi Dayan. Android keystore stack buffer overflow - cve-2014–3100, 2014.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *Proceedings of the 19th Network and Distributed Security Symposium*, NDSS '12. Internet Society, 2012.

[Hem04] Ludger Hemme. A differential fault attack against early rounds of (triple-)des. In *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 254–267. Springer-Verlag, 2004.

[HHF+05] Hermann Härtig, Michael Hohmuth, Norman Feske, Christian Helmuth, Adam Lackorzynski, Frank Mehnert, and Michael Peter. The nizza secure-system architecture. In *2005 International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 10 pp.–, 2005.

[HHF09] Ralf Hund, Thorsten Holz, and Felix C. Freiling. Return-oriented rootkits: Bypassing kernel code integrity protection mechanisms. In *Proceedings of the 18th Conference on USENIX Security Symposium*, SSYM '09, pages 383–398. USENIX Association, 2009.

[HKD+13] Owen S. Hofmann, Sangman Kim, Alan M. Dunn, Michael Z. Lee, and Emmett Witchel. Inktag: Secure applications on an untrusted operating system. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 265–278. ACM, 2013.

[HM08] Markus C. Huebscher and Julie A. McCann. A survey of autonomic computing–degrees, models, and applications. *ACM Computer Survey*, 40(3):7:1–7:28, August 2008.

[HOL16] HOL4. Hol interactive theorem prover, January 2016. https://hol-theorem-prover.org/.

[Hot16] Hotdog Games. Tee checker, January 2016. https://play.google.com/store/apps/details?id=com.ifihada.teechecker&hl=en.

[HSG11] Nathaniel Husted, Hassen Saïdi, and Ashish Gehani. Smartphone security limitations: Conflicting traditions. In *Proceedings of the 2011 Workshop on Governance of Technology, Information, and Policies*, GTIP '11, pages 5–12. ACM, 2011.

[HSH+08] Joo-Young Hwang, Sang-Bum Suh, Sung-Kwan Heo, Chan-Ju Park, Jae-Min Ryu, Seong-Yeol Park, and Chul-Ryun Kim. Xen on arm: System virtualization using xen hypervisor for arm-based secure mobile phones. In *Proceedings of the 5th IEEE Consumer Communications and Networking Conference*, CCNC '08, pages 257–261. IEEE, 2008.

[HSH+09] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold-boot attacks on encryption keys. *Communications of the ACM - Security in the Browser*, 52(5):91–98, May 2009.

[HWF15] Daniel Hein, Johannes Winter, and Andreas Fitzek. Secure block device – secure, flexible, and efficient data storage for arm trustzone systems. In *Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA - Volume 01*, TRUSTCOM '15, pages 222–229. IEEE, 2015.

[IBM01] IBM. Autonomic computing: Ibm's perspective on the state of information technology, October 2001. `http://people.scs.carleton.ca/~soma/biosec/readings/autonomic_computing.pdf`.

[IDC16a] IDC. Smartphone os market share, 2015 q2, 2016. `http://www.idc.com/prodserv/smartphone-os-market-share.jsp`.

[IDC16b] IDC. Worldwide smartphone forecast update, 2016–2020, 2016. `http://www.idc.com/getdoc.jsp?containerId=US41515416`.

[IK03] Tetsu Iwata and Kaoru Kurosawa. Omac: One-key cbc mac. In *Fast Software Encryption – FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 129–153. Springer-Verlag, 2003.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161. Springer-Verlag, 2003.

[Inf07] Information Assurance Directorate. U.s. government protection profile for separation kernels in environments requiring high robustness – version 1.03, June 2007. `https://www.niap-ccevs.org/pp/pp_skpp_hr_v1.03.pdf`.

[ISO02] ISO/IEC JTC 1/SC 6. Information technology – asn.1 encoding rules: Specification of basic encoding rules (ber), canonical encoding rules (cer) and distinguished encoding rules (der). Technical report, International Organization for Standardization, December 2002.

[ISO06] ISO/IEC JTC 1/SC 27. Information technology – security techniques – modes of operation for an n-bit block cipher. Technical report, International Organization for Standardization, February 2006.

[ISO13] ISO/IEC JTC 1/SC 17. Identification cards – integrated circuit cards – part 4: Organization, security and commands for interchange. Technical report, International Organization for Standardization, April 2013.

[ISO16] ISO/IEC JTC 1/SC 27. Information technology – security techniques – information security management systems – overview and vocabulary. Technical report, International Organization for Standardization, February 2016.

[Jae08] Trent Jaeger. *Operating system security.* Synthesis lectures on information security, privacy and trust. Morgan & Claypool Publishers, 2008.

[Jav14] Java Card Forum. Java card forum returns to hong kong to hold open day, March 2014. `https://javacardforum.com/activities/in-the-news/`.

[JKK+15] Jinsoo Jang, Sunjune Kong, Minsu Kim, Daegyeong Kim, and Brent Byunghoon Kang. Secret: Secure channel between rich execution environment and trusted execution environment. In *Proceedings of the 22nd Annual Network and Distributed System Security Symposium*, NDSS '15. Internet Society, 2015.

[JMV02] Antoine Joux, Gwenaëlle Martinet, and Frédéric Valette. Blockwise-adaptive attackers: Revisiting the (in)security of some provably secure encryption models: Cbc, gem, iacbc. In *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 17–30. Springer-Verlag, 2002.

[Kal00] Burt Kaliski. Pkcs #5: Password-based cryptography specification, version 2.0. RFC 2898, RFC Editor, September 2000. `http://www.rfc-editor.org/rfc/rfc2898.txt`.

[KBC97] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. Hmac: Keyed-hashing for message authentication. RFC 2104, RFC Editor, February 1997. `http://www.rfc-editor.org/rfc/rfc2104.txt`.

[KEAR09] Kari Kostiainen, Jan-Erik Ekberg, N. Asokan, and Aarne Rantala. On-board credentials with open provisioning. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, pages 104–115. ACM, 2009.

[KEH+09] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 207–220. ACM, 2009.

[KFM04] Maxwell N. Krohn, Michael J. Freedman, and David Mazieres. On-the-fly verification of rateless erasure codes for efficient content distribution. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 226–240. IEEE, May 2004.

[KH14] Nathan Keltner and Charles Holmes. Here be dragons: vulnerabilities in trustzone, August 2014. `http://atredispartners.blogspot.jp/2014/08/here-be-dragons-vulnerabilities-in.html`.

[KK13] Vladimir Kolesnikov and Ranjit Kumaresan. Improved ot extension for transferring short secrets. In *Advances in Cryptology – CRYPTO 2013*, volume 8043 of *Lecture Notes in Computer Science*, pages 54–70. Springer-Verlag, 2013.

[KKMN10] Kei Kawamorita, Ryouta Kasahara, Yuuki Mochizuki, and Kenichiro Noguchi. Application of formal methods for designing a separation kernel for embedded systems. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 4(8):150–158, August 2010.

[KL15] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography.* Chapman & Hall Book, 2015. second edition.

[KM07] Neal Koblitz and Alfred J. Menezes. Another look at "provable security". *Journal of Cryptology*, 20(1):3–37, January 2007.

[KM16] Matt Kaufmann and J Strother Moore. Acl2 version 7.2, January 2016. http://www.cs.utexas.edu/users/moore/acl2/.

[KPK14] Vasileios P. Kemerlis, Michalis Polychronakis, and Angelos D. Keromytis. Ret2dir: Rethinking kernel isolation. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC '14, pages 957–972. USENIX Association, 2014.

[Kra01] Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is ssl?). In *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 310–331. Springer-Verlag, 2001.

[Kun16] Sravan Kundojjala. Baseband market share tracker q4 2015: Mediatek, samsung lsi take lte share from qualcomm, March 2016. https://www.strategyanalytics.com/access-services/components/rf-and-wireless/market-data/report-detail/baseband-market-share-tracker-q4-2015-mediatek-samsung-lsi-take-lte-share-from-qualcomm#.V39wX0bpLws.

[Lag16] Laginimaineb. Trustzone kernel privilege escalation (cve-2016-2431), June 2016. http://bits-please.blogspot.fr/2016/06/trustzone-kernel-privilege-escalation.html.

[Lam74] Butler W. Lampson. Protection. *SIGOPS Operating Systems Review*, 8(1):18–24, January 1974.

[LC14] Dongtao Liu and Landon P. Cox. Veriui: Attested login for mobile devices. In *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*, HotMobile '14, pages 7:1–7:6. ACM, 2014.

[LHSB13] Li Li, Dijiang Huang, Zhidong Shen, and Samia Bouzefrane. A cloud based dual-root trust model for secure mobile online transactions. In *Proceedings of the 2013 IEEE Wireless Communications and Networking Conference*, WCNC '13, pages 4404–4409. IEEE, 2013.

[LHW15] Christian Lesjak, Daniel Hein, and Johannes Winter. Hardware-security technologies for industrial iot: Trustzone and security controller. In *Proceedings of the 41st Annual Conference of the IEEE Industrial Electronics Society*, IECON '15, pages 2589–2595. IEEE, 2015.

[LLL+11] Matthias Lange, Steffen Liebergeld, Adam Lackorzynski, Alexander Warg, and Michael Peter. L4android: A generic operating system framework for secure smartphones. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, pages 39–50. ACM, 2011.

[LMH+14] Wenhao Li, Mingyang Ma, Jinchen Han, Yubin Xia, Binyu Zang, Cheng-Kang Chu, and Tieyan Li. Building trusted path on untrusted device drivers for mobile devices. In *Proceedings of 5th Asia-Pacific Workshop on Systems*, APSys '14, pages 8:1–8:7. ACM, 2014.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[LPMS13] Mariantonietta La Polla, Fabio Martinelli, and Daniele Sgandurra. A survey on security for mobile devices. *IEEE Communications Surveys Tutorials*, 15(1):446–471, First quarter 2013.

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer-Verlag, 2002.

[LSW10] Hans Löhr, Ahmad-Reza Sadeghi, and Marcel Winandy. Patterns for secure boot and secure storage in computer systems. In *Proceedings of the 5th International Conference on Availability, Reliability, and Security*, ARES '10, pages 569–573. IEEE, 2010.

[LSWR12] He Liu, Stefan Saroiu, Alec Wolman, and Himanshu Raj. Software abstractions for trusted sensors. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, pages 365–378. ACM, 2012.

[LTH03] David Lie, Chandramohan A. Thekkath, and Mark Horowitz. Implementing an untrusted operating system on trusted hardware. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 178–192. ACM, 2003.

[MAM15] Konstantinos Markantonakis, Raja Naeem Akram, and Mehari G. Msgna. Secure and trusted application execution on embedded devices. In *Innovative Security Solutions for Information Technology and Communications – SECITC 2015*, volume 9522 of *Lecture Notes in Computer Science*, pages 3–24. Springer-Verlag, 2015.

[Mar98] Constantinos Markantonakis. The case for a secure multi-application smart card operating system. In *Proc. of the First International Workshop on Information Security*, ISW '97, pages 188–197. Springer-Verlag, 1998.

[MBOS16] Fadi Mohsen, Emmanuel Bello-Ogunu, and Mohamed Shehab. Investigating the keylogging threat in android - user perspective. In *Proceedings of the Second International Conference on Mobile and Secure Services (MobiSecServ)*, MobiSecServ '16, pages 1–5. IEEE, 2016.

[McG06] Gary McGraw. *Software Security: Building Security In*. Addison-Wesley Professional, 2006.

[MD13] Anmol Misra and Abhishek Dubey. *Android Security: Attacks and Defenses*. Taylor & Francis, 2013.

[MDNA15] Brian McGillion, Tanel Dettenborn, Thomas Nyman, and N. Asokan. Open-tee – an open virtual trusted execution environment. In *Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA - Volume 01*, TRUSTCOM '15, pages 400–407. IEEE, 2015.

[MFAM16] Mehari G. Msgna, Houda Ferradi, Raja Naeem Akram, and Konstantinos Markantonakis. Secure application execution in mobile devices. In *The New Codebreakers*, volume 9100 of *Lecture Notes in Computer Science*, pages 417–438. Springer-Verlag, 2016.

[MGBF14] Benjamin Mood, Debayan Gupta, Kevin Butler, and Joan Feigenbaum. Reuse it or lose it: More efficient secure computation through reuse of encrypted values. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 582–596. ACM, 2014.

[MGC+16] Benjamin Mood, Debayan Gupta, Henry Carter, Kevin R. B. Butler, and Patrick Traynor. Frigate: A validated, extensible, and efficient compiler and interpreter for secure computation. In *Proceedings of the 2016 IEEE European Symposium on Security and Privacy*, EuroSP '16, pages 112–127. IEEE, March 2016.

[MGS11] Collin Mulliner, Nico Golde, and Jean-Pierre Seifert. Sms of death: From analyzing to attacking mobile phones on a large scale. In *Proceedings of the 20th USENIX Conference on Security*, SEC '11, pages 24–24. USENIX Association, 2011.

[Mit03] Chris Mitchell. *Security for Mobility.* IEEE Press, 2003.

[Mit05a] Chris J. Mitchell. Cryptanalysis of two variants of pcbc mode when used for message integrity. In *Information Security and Privacy*, volume 3574 of *Lecture Notes in Computer Science*, pages 560–571. Springer-Verlag, 2005.

[Mit05b] Chris J. Mitchell. Error oracle attacks on cbc mode: Is there a future for cbc mode encryption? In *Information Security – ISC '05*, volume 3650 of *Lecture Notes in Computer Science*, pages 244–258. Springer-Verlag, 2005.

[Mit13] Chris J. Mitchell. Analysing the iobc authenticated encryption mode. In *Information Security and Privacy*, volume 7959 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, 2013.

[Mit16] Chris J. Mitchell. On the security of 2-key triple DES. CoRR, abs/1602.06229, 2016.

[MKS+14] Claudio Marforio, Nikolaos Karapanos, Claudio Soriente, Kari Kostiainen, and Srdjan Čapkun. Smartphones as practical and secure location verification tokens for payments. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium*, NDSS '14. Internet Society, 2014.

[MM09] Collin Mulliner and Charlie Miller. Injecting sms messages into smart phones for security analysis. In *Proceedings of the 3rd USENIX Conference on Offensive Technologies*, WOOT '09, pages 5–5. USENIX Association, 2009.

[MMB+13] Toby Murray, Daniel Matichuk, Matthew Brassil, Peter Gammie, Timothy Bourke, Sean Seefried, Corey Lewis, Xin Gao, and Gerwin Klein. sel4: From general purpose to a proof of information flow enforcement. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 415–429. IEEE, 2013.

[MPP+08] Jonathan M. McCune, Bryan J. Parno, Adrian Perrig, Michael K. Reiter, and Hiroshi Isozaki. Flicker: An execution infrastructure for tcb minimization. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, Eurosys '08, pages 315–328. ACM, 2008.

[MS13] Tilo Müller and Michael Spreitzenbarth. Frost: Forensic recovery of scrambled telephones. In *Applied Cryptography and Network Security – ACNS 2013*, volume 7954 of *Lecture Notes in Computer Science*, pages 373–388. Springer-Verlag, 2013.

[Mul06] Collin Richard Mulliner. *Security of Smart Phones.* University of California, Santa Barbara, 2006.

[MV03] David A. McGrew and John Viega. Flexible and efficient message authentication in hardware and software. Manuscript, 2003.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 448–457. Society for Industrial and Applied Mathematics, 2001.

[NRS14] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering generic composition. In *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 257–274. Springer-Verlag, 2014.

[Nvi14] Nvidia. Tlk: A foss stack for secure hardware tokens, 2014. https://www.w3.org/2012/webcrypto/webcrypto-next-workshop/papers/webcrypto2014_submission_25.pdf.

[NXP16] NXP. Smartmx2-p60, 2016. http://www.nxp.com/products/identification-and-security/smart-card-ics/smartmx2-p60:MC_71471.

[OKK+12] Soo-Cheol Oh, KwangWon Koh, Chei-Yol Kim, KangHo Kim, and SeongWoon Kim. Acceleration of dual os virtualization in embedded systems. In *Proceedings of the 7th International Conference on Computing and Convergence Technology*, ICCCT '12, pages 1098–1101, 2012.

[OMT09] OMTP Limited. Advanced trusted environment: Omtp tr1 – v 1.1, May 2009. http://www.omtp.org/OMTP_Advanced_Trusted_Environment_OMTP_TR1_v1_1.pdf.

[Ora12] Oracle. Java card protection profile – closed configuration. Common Criteria for Information Technology Security Evaluation, December 2012. Certification Report: ANSSI-CC-PP-2010/07.

[oS77] National Bureau of Standards. Data encryption standard. Federal Information Processing Standard (FIPS), January 1977. Publication 46.

[oSN85] National Institute of Standards and Technology (NIST). omputer data authentication. Federal Information Processing Standard (FIPS), May 1985. Publication 113.

[oSN01] National Institute of Standards and Technology (NIST). Advanced encryption standard. Federal Information Processing Standard (FIPS), November 2001. Publication 197.

[PCSL08] Bryan D. Payne, Martim Carbone, Monirul Sharif, and Wenke Lee. Lares: An architecture for secure active monitoring using virtualization. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, SP '08, pages 233–247. IEEE, 2008.

[PH07] Nick L. Petroni, Jr. and Michael Hicks. Automated detection of persistent kernel control-flow attacks. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 103–115. ACM, 2007.

[PHY05] PHYS ORG. Stmicroelectronics, orange, and trusted logic to demonstrate secure mobile phone and payment application, February 2005. http://phys.org/news/2005-02-stmicroelectronics-orange-logic-mobile-payment.html.

[PKR+13] Niels Penneman, Danielius Kudinskas, Alasdair Rawsthorne, Bjorn De Sutter, and Koen De Bosschere. Formal virtualization requirements for the arm architecture. *Journal of Systems Architecture*, 59(3):144–154, March 2013.

[PLD+11] Bryan Parno, Jacob R. Lorch, John R. Douceur, James Mickens, and Jonathan M. McCune. Memoir: Practical state continuity for protected modules. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, pages 379–394. IEEE, 2011.

[Pre98] Bart Preneel. Cryptographic primitives for information authentication – state of the art. In *State of the Art in Applied Cryptography*, volume 1528 of *Lecture Notes in Computer Science*, pages 49–104. Springer-Verlag, 1998.

[PS10] Vaibhav Ranchhoddas Pandya and Mark Stamp. iphone security analysis. *Journal of Information Security*, 1(2):74–87, 2010.

[PS12] Martin Pirker and Daniel Slamanig. A framework for privacy-preserving mobile payment on security enhanced arm trustzone platforms. In *Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, TRUSTCOM '12, pages 1155–1160. IEEE, 2012.

[PSW12] Martin Pirker, Daniel Slamanig, and Johannes Winter. Practical privacy preserving cloud resource-payment for constrained clients. In *Privacy Enhancing Technologies – PETS 2012*, volume 7384 of *Lecture Notes in Computer Science*, pages 201–220. Springer-Verlag, 2012.

[PW12] Kenneth G. Paterson and Gaven J. Watson. Authenticated-encryption with padding: A formal security treatment. In *Cryptography and Security: From Theory to Applications*, volume 6805 of *Lecture Notes in Computer Science*, pages 83–107. Springer-Verlag, 2012.

[Qua16a] Qualcomm. Snapdragon mobile security, 2016. `https://www.qualcomm.com/products/snapdragon/security`.

[Qua16b] Qualcomm. Snapdragon modems, 2016. `https://www.qualcomm.com/products/snapdragon/modems`.

[Rab79] Michael O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical Report MIT-LCS-TR-212, Massachusetts Institute of Technology (Cambridge, MA US), January 1979.

[Rab81] Michael O. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken Computation Lab, Harvard University, 1981.

[RD05] John Regehr and Usit Duongsaa. Preventing interrupt overload. In *Proceedings of the 2005 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, LCTES '05, pages 50–58. ACM, 2005.

[RE10] Wolfgang Rankl and Wolfgang Effing. *Smart Card Handbook*. John Wiley & Sons, Inc., 4th edition, June 2010.

[Ric10] Raymond J. Richards. *Modeling and Security Analysis of a Commercial Real-Time Operating System Kernel*, pages 301–322. Springer US, 2010.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 98–107. ACM, 2002.

[Rog04a] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes ocb and pmac. In *Advances in Cryptology – ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer-Verlag, 2004.

[Rog04b] Phillip Rogaway. Nonce-based symmetric encryption. In *Fast Software Encryption – FSE 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 348–358. Springer-Verlag, 2004.

[Rog11] Phillip Rogaway. Evaluation of some blockcipher modes of operation. Technical report, Cryptography Research and Evaluation Committees (CRYPTREC) for the Government of Japan, 2011.

[Ros13] Dan Rosenberg. Unlocking the motorola bootloader, August 2013. `http://blog.azimuthsecurity.com/2013/04/unlocking-motorola-bootloader.html`.

[Ros14] Dan Rosenberg. Reflections on trusting trustzone. In *Blackhat USA 2014*. BlackHat, 2014.

[RPV15] Anil Kumar Reddy, Periyasamy Paramasivam, and Prakash Babu Vemula. Mobile secure data protection using emmc rpmb partition. In *Proceedings of the 2015 International Conference on Computing and Network Communications*, CoCoNet '15, pages 946–950. IEEE, 2015.

[RSW+16] Himanshu Raj, Stefan Saroiu, Alec Wolman, Ronald Aigner, Jeremiah Cox, Paul England, Chris Fenner, Kinshuman Kinshumann, Jork Loeser, Dennis Mattoon, Magnus Nystrom, David Robinson, Rob Spiger, Stefan Thom, and David Wooten. ftpm: A software-only implementation of a tpm chip. In *Proceedings of the 25th USENIX Security Symposium*, SSYM '16. USENIX Association, 2016.

[Rus81] John M. Rushby. Design and verification of secure systems. In *Proceedings of the Eighth ACM Symposium on Operating Systems Principles*, SOSP '81, pages 12–21. ACM, 1981.

[SA16a] Mohamed Sabt and Mohammed Achemlal. Garbling the trust: Trusted execution environment based on garbled circuits. unpublished, 2016.

[SA16b] Mohamed Sabt and Mohammed Achemlal. Procédé de protection d'un terminal mobile contre des attaques, April 2016. WO Patent 051059.

[SA16c] Mohamed Sabt and Mohammed Achemlal. ustore: Resource-efficient uicc-based android keystore in practice. unpublished, 2016.

[SAA16] Mohamed Sabt, Mouhannad Alattar, and Mohammed Achemlal. Procédé de sécurisation de transactions sans contact, June 2016. WO Patent 102831.

[SAB15a] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. The dual-execution-environment approach: Analysis and comparative evaluation. In *ICT Systems Security and Privacy Protection – IFIP SEC '15*, volume 455 of *IFIP Advances in Information and Communication Technology*, pages 557–570. Springer-Verlag, 2015.

[SAB15b] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Over-the-internet: Efficient remote content management for secure elements in mobile devices. In *Proceedings of the first International Conference on Mobile and Security Services*, MobiSecServ '15, pages 1–5. IEEE, 2015.

[SAB15c] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Towards integrating trusted execution environment into embedded autonomic systems. In *Proceedings of the 2015 IEEE International Conference on Autonomic Computing*, ICAC '15, pages 165–166. IEEE, 2015.

[SAB15d] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted execution environment: What it is, and what it is not. In *Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA - Volume 01*, TRUSTCOM '15, pages 57–64. IEEE, 2015.

[Sab16] Mohamed Sabt. Trusted execution environment: Trusted model, architecture overview, challenges and future directions. In *Proceedings of the Second International Conference on Mobile and Security Services*, MobiSecServ '16, pages 1–1. IEEE, 2016.

[Sam16] Samsung. Samsung exynos processor, 2016. `http://www.samsung.com/semiconductor/minisite/Exynos/w/solution/overview/`.

[SBTC15] Sampath Srinivas, Dirk Balfanz, Eric Tiffany, and Alexei Czeskis. Universal 2nd factor (u2f) overview. Technical report, FIDO Alliance, May 2015.

[SCG+03] G. Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. Aegis: Architecture for tamper-evident and tamper-resistant processing. In *Proceedings of the 17th Annual International Conference on Supercomputing*, ICS '03, pages 160–171. ACM, 2003.

[Sen13] SensePost. A software level analysis of trustzone os and trustlets in samsung galaxy phone, June 2013. `https://www.sensepost.com/blog/2013/a-software-level-analysis-of-trustzone-os-and-trustlets-in-samsung-galaxy-phone/`.

[Seq16a] Sequitur Labs Inc. Coretee - foundational security for iot systems, 2016. `http://www.sequiturlabs.com/coretee/`.

[Seq16b] Sequitur Labs Inc. Sequitur labs collaborates with linaro to lower barriers to iot security education for raspberry pi maker community, June 2016. `http://www.sequiturlabs.com/media_portfolio/sequitur-labs-collaborates-with-linaro-to-lower-barriers-to-iot-security-education-for-raspberry-pi-maker-community/`.

[SFK+10] Asaf Shabtai, Yuval Fledel, Uri Kanonov, Yuval Elovici, Shlomi Dolev, and Chanan Glezer. Google android: A comprehensive security assessment. *IEEE Security Privacy*, 8(2):35–44, March 2010.

[SGB+16] Junaid Shuja, Abdullah Gani, Kashif Bilal, Atta Ur Rehman Khan, Sajjad A. Madani, Samee U. Khan, and Albert Y. Zomaya. A survey of mobile device virtualization: Taxonomy and state of the art. *ACM Computer Survey*, 49(1):1:1–1:36, April 2016.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems*. *Bell System Technical Journal*, 28(4):656–715, 1949.

[SHS+15] Ebrahim M. Songhori, Siam U. Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, pages 411–428. IEEE, May 2015.

[SHT10] Daniel Sangorrín, Shinya Honda, and Hiroaki Takada. Dual operating system architecture for real-time embedded systems. In *Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT)*, OSPERT '10, pages 6–15, 2010.

[SHT12] Daniel Sangorrín, Shinya Honda, and Hiroaki Takada. Integrated scheduling for a reliable dual-os monitor. *IPSJ Transactions on Advanced Computing Systems (ACS)*, 5(2):99–110, March 2012.

[SHT13] Daniel Sangorrín, Shinya Honda, and Hiroaki Takada. Reliable and efficient dual-os communications for real-time embedded virtualization. *Information and Media Technologies*, 8(1):1–17, March 2013.

[Sie16] SierraWare. Sierratee trusted execution environment, 2016. `http://www.sierraware.com/open-source-ARM-TrustZone.html`.

[SIM15] SIMalliance. Open mobile api – archive, 2015. `http://simalliance.org/se/se-technical-releases/simalliance-open-mobile-api-archive/`.

[SLPR15] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 213–226. ACM, 2015.

[SLQP07] Arvind Seshadri, Mark Luk, Ning Qu, and Adrian Perrig. Secvisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity oses. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 335–350. ACM, 2007.

[SLS+05] Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, SOSP '05, pages 1–16. ACM, 2005.

[Sol16] Solacia. Securitee, 2016. `http://www.sola-cia.com/en/securiTee/product.asp`.

[SP12] Raoul Strackx and Frank Piessens. Fides: Selectively hardening software application components against kernel-level or process-level malware. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 2–13. ACM, 2012.

[SRSW14] Nuno Santos, Himanshu Raj, Stefan Saroiu, and Alec Wolman. Using arm trustzone to build a trusted language runtime for mobile applications. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 67–80. ACM, 2014.

[SSW+15] He Sun, Kun Sun, Yuewu Wang, Jiwu Jing, and Haining Wang. Trustice: Hardware-assisted isolated computing environments on mobile devices. In *Proceedings of the 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '15, pages 367–378. IEEE, 2015.

[SSWJ15] He Sun, Kun Sun, Yuewu Wang, and Jiwu Jing. Trustotp: Transforming smartphones into secure one-time password tokens. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 976–988. ACM, 2015.

[ST16a] Mohamed Sabt and Jacques Traoré. Breaking into the keystore: A practical forgery attack against android keystore. In *Computer Security – ESORICS 2016 – Part 2*, volume 9879 of *Lecture Notes in Computer Science*, pages 531–548. Springer-Verlag, 2016.

[ST16b] Mohamed Sabt and Jacques Traoré. Cryptanalysis of globalplatform secure channel protocols. In *Security Standardisation Research – SSR 2016*, volume 10074 of *Lecture Notes in Computer Science*, pages 62–91. Springer-Verlag, 2016.

[Sta16] Statista. Number of available applications in the google play store from december 2009 to february 2016, 2016. `http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/`.

[STCT15] Sufatrio, Darell J. J. Tan, Tong-Wei Chua, and Vrizlynn L. L. Thing. Securing android: A survey, taxonomy, and challenges. *ACM Computer Survey*, 47(4):58:1–58:45, May 2015.

[Ste13] Maximilian Stein. Mobile devices as secure eid reader using trusted execution environments. In *Proceedings of the 2013 Open Identity Summit*, OID '13, pages 11–19. Springer-Verlag, 2013.

[TEA11] Sandeep Tamrakar, Jan-Erik Ekberg, and N. Asokan. Identity verification schemes for public transport ticketing with nfc phones. In *Proceedings of the Sixth ACM Workshop on Scalable Trusted Computing*, STC '11, pages 37–48. ACM, 2011.

[TFH+14] Peter Teufl, Andreas Gregor Fitzek, Daniel Hein, Alexander Marsalek, Alexander Oprisnik, and Thomas Zefferer. Android encryption systems. In *International Conference on Privacy & Security in Mobile Systems*, 2014.

[Tru16] Trustonic. Kinibi trusted execution environment (tee), 2016. `https://www.trustonic.com/products/kinibi`.

[TWP13] Ronald Tögl, Johannes Winter, and Martin Pirker. A path towards ubiquitous protection of media. In *Proceedings of the Workshop on Web Applications and Secure Hardware (WASH '13)*, volume 1011 of *CEUR Workshop Proceedings*, pages 32–38. Technical University of Aachen, 2013.

[UA16] Pascal Urien and Xavier Aghina. Secure mobile payments based on cloud services: Concepts and experiments. In *Proceedings of the 2nd IEEE International Conference on Big Data Security on Cloud, IEEE International Conference on High Performance and Smart Computing, and IEEE International Conference on Intelligent Data and Security*, BigDataSecurity-HPSC-IDS '16, pages 333–338. IEEE, April 2016.

[VLB+05] Mark W. Vanfleet, Jahn A. Luke, William R. Beckwith, Carol Taylor, Ben Calloni, and Gordon Uchenick. Mils: Architecture for high-assurance embedded computing. *CrossTalk: Journal of Defence Software Engineering*, 18(8):12–16, February 2005.

[Vol99] Heribert Vollmer. *Introduction to Circuit Complexity*. Springer-Verlag New York, Inc., 1st edition, 1999.

[VOZ+12] Amit Vasudevan, Emmanuel Owusu, Zongwei Zhou, James Newsome, and Jonathan M. McCune. Trustworthy execution on mobile devices: What security properties can my mobile platform give me? In *Trust and Trustworthy Computing – TRUST 2012*, volume 7344 of *Lecture Notes in Computer Science*, pages 159–178. Springer-Verlag, 2012.

[VPKE14] Sebastian Vogl, Jonas Pfoh, Thomas Kittel, and Claudia Eckert. Persistent data-only malware: Function hooks without code. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium*, NDSS '14. Internet Society, 2014.

[vRDP13] Roland van Rijswijk-Deij and Erik Poll. Using trusted execution environments in two-factor authentication: comparing approaches. In *Proceedings of the 2013 Open Identity Summit*, OID '13, pages 20–31. Springer-Verlag, 2013.

[Wei12] Ralf-Philipp Weinmann. Baseband attacks: Remote exploitation of memory corruptions in cellular protocol stacks. In *Proceedings of the 6th USENIX Conference on Offensive Technologies*, WOOT '12, pages 2–2. USENIX Association, 2012.

[WFM+07] Peter Wilson, Alexandre Frey, Tom Mihm, Danny Kershaw, and Tiago Alves. Implementing embedded security on dual-virtual-cpu systems. *IEEE Design & Test*, 24(6):582–591, November 2007.

[WHCE05] Wan Huzaini Wan Hussin, Paul Coulton, and Reuben Edwards. Mobile ticketing system employing trustzone technology. In *Proceedings of the 2005 International Conference on Mobile Business*, ICMB '05, pages 651–654. IEEE, 2005.

[WHEC06] Wan Huzaini Wan Hussin, Reuben Edwards, and Paul Coulton. E-pass using drm in symbian v8 os and trustzone: Securing vital data on mobile devices. In *Proceedings of the International Conference on Mobile Business*, ICMB '06, pages 14–18. IEEE, 2006.

[WHF03] Doug Whiting, Russ Housley, and Niels Ferguson. Counter with cbc-mac (ccm). RFC 3610, RFC Editor, September 2003. `http://www.rfc-editor.org/rfc/rfc3610.txt`.

[WHW+04] Steve R. White, James E. Hanson, Ian Whalley, David M. Chess, and Jeffrey O. Kephart. An architectural approach to autonomic computing. In *Proceedings of the First IEEE International Conference on Autonomic Computing*, ICAC '04, pages 2–9. IEEE, 2004.

[Wil13] Lee Wilson. The tcg dynamic root for trusted measurement, June 2013. `https://www.trustedcomputinggroup.org/wp-content/uploads/DRTM-Specification-Overview_June2013.pdf`.

[Win13] Johannes Winter. Experimental version of qemu with basic support for arm trustzone (security extensions), June 2013. `https://github.com/jowinter/qemu-trustzone`.

[Woo10] Ben Woods. Researchers expose android webkit browser exploit, November 2010. `http://www.zdnet.com/article/researchers-expose-android-webkit-browser-exploit/`.

[WWPT12] Johannes Winter, Paul Wiegele, Martin Pirker, and Ronald Tögl. A flexible software development and emulation framework for arm trustzone. In *Trusted Systems – INTRUST 2011*, volume 7222 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2012.

[XLS+15] Wen Xu, Juanru Li, Junliang Shu, Wenbo Yang, Tianyi Xie, Yuanyuan Zhang, and Dawu Gu. From collision to exploitation: Unleashing use-after-free vulnerabilities in linux kernel. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 414–425. ACM, 2015.

[XN15] Christos Xenakis and Christoforos Ntantogian. Attacking the baseband modem of mobile phones to breach the users' privacy and network security. In *Proceedings of the 7th International Conference on Cyber Conflict: Architectures in Cyberspace*, CyCon '15, pages 231–244. IEEE, 2015.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164. IEEE, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, SFCS '86, pages 162–167. IEEE, 1986.

[YYZ+16] Bo Yang, Kang Yang, Zhenfeng Zhang, Yu Qin, and Dengguo Feng. Aep-m: Practical anonymous e-payment for mobile devices using arm trustzone and divisible e-cash (full version). Cryptology ePrint Archive, Report 2016/494, 2016.

[ZDH+13] Xiaoyong Zhou, Soteris Demetriou, Dongjing He, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, Carl A. Gunter, and Klara Nahrstedt. Identity, location, disease and more: Inferring your secrets from android public resources. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 1017–1028. ACM, 2013.

[Zor16] John Zorabedian. Mobile security updates are a mess. the fcc and ftc want to know why, May 2016. `https://nakedsecurity.sophos.com/2016/05/11/mobile-security-updates-are-a-mess-the-fcc-and-ftc-want-to-know-why/`.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In *Advances in Cryptology – EUROCRYPT 2015*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250. Springer-Verlag, 2015.

[ZWC+13] Chao Zhang, Tao Wei, Zhaofeng Chen, Lei Duan, Laszlo Szekeres, Stephen McCamant, Dawn Song, and Wei Zou. Practical control flow integrity and randomization for binary executables. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 559–573. IEEE, 2013.

[ZWWJ15] Yajin Zhou, Lei Wu, Zhi Wang, and Xuxian Jiang. Harvesting developer credentials in android apps. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '15, pages 23:1–23:12. ACM, 2015.

[ZZH+14] Shijun Zhao, Qianying Zhang, Guangyao Hu, Yu Qin, and Dengguo Feng. Providing root of trust for arm trustzone using on-chip sram. In *Proceedings of the 4th International Workshop on Trustworthy Embedded Devices*, TrustED '14, pages 25–36. ACM, 2014.

Mohamed Sabt

Outsmarting Smartphones – Trust Based on Provable Security and Hardware Primitives in Smartphones Architectures

Summary:

The landscape of mobile devices has been changed with the introduction of smartphones. Since their advent, smartphones have become almost vital in the modern world. This has spurred many service providers to propose access to their services via mobile applications. Despite such big success, the use of smartphones for sensitive applications has not become widely popular. The reason behind this is that users, being increasingly aware about security, do not trust their smartphones to protect sensitive applications from attackers. The goal of this thesis is to strengthen users trust in their devices. We cover this *trust problem* with two complementary approaches: provable security and hardware primitives.

In the first part, our goal is to demonstrate the limits of the existing technologies in smartphones architectures. To this end, we analyze two widely deployed systems in which careful design was applied in order to enforce their security guarantee: the Android KeyStore, which is the component shielding users cryptographic keys in Android smartphones, and the family of Secure Channel Protocols (SCPs) defined by the GlobalPlatform consortium. Our study relies on the paradigm of provable security. Despite being perceived as rather theoretical and abstract, we show that this tool can be handily used for real-world systems to find security vulnerabilities. This shows the important role that can play provable security for *trust* by being able to formally prove the absence of security flaws or to identify them if they exist.

The second part focuses on complex systems that cannot cost-effectively be formally verified. We begin by investigating the dual-execution-environment approach. Then, we consider the case when this approach is built upon some particular hardware primitives, namely the ARM TrustZone, to construct the so-called Trusted Execution Environment (TEE). Finally, we explore two solutions addressing some of the TEE limitations. First, we propose a new TEE architecture that protects its sensitive data even when the secure kernel gets compromised. This relieves service providers of fully trusting the TEE issuer. Second, we provide a solution in which TEE is used not only for execution protection, but also to guarantee more elaborated security properties (i.e. self-protection and self-healing) to a complex software system like an OS kernel.

Keywords: Smartphone Security, Provable Security, Android KeyStore, GlobalPlatform, SCP, ARM TrustZone, TEE, Garbled Circuits, Autonomic System