Department of Computer Science and Engineering

On-board Credentials: An Open Credential Platform for Mobile Devices

Kari Kostiainen

Aalto University

DOCTORAL DISSERTATIONS

Aalto University publication series
DOCTORAL DISSERTATIONS 49/2012

On-board Credentials: An Open Credential Platform for Mobile Devices

Kari Kostiainen

Doctoral dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the School of Science for public examination and debate in Auditorium TU1 at the Aalto University School of Science (Espoo, Finland) on the 25th of May 2012 at 12 noon.

Aalto University
School of Science
Department of Computer Science and Engineering

Supervisor
Professor Tuomas Aura

Instructor
Dr. N. Asokan, Nokia Research Center, Helsinki

Preliminary examiners
Assistant Professor Mohammad Mannan, Concordia University, Canada
Dr. Jonathan M. McCune, Carnegie Mellon University, USA

Opponent
Professor Chris Mitchell, Royal Holloway, University of London, UK

Aalto University publication series
DOCTORAL DISSERTATIONS 49/2012

ISBN 978-952-60-4597-9 (printed)
ISBN 978-952-60-4598-6 (pdf)
ISSN-L 1799-4934
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)

Unigrafia Oy
Helsinki 2012

Finland

The dissertation can be read at http://lib.tkk.fi/Diss/

441 697
Printed matter

Aalto University

Abstract

Aalto University, P.O. Box 11000, FI-00076 Aalto www.aalto.fi

Author
Kari Kostiainen

Name of the doctoral dissertation
On-board Credentials: An Open Credential Platform for Mobile Devices

Publisher School of Science

Unit Department of Computer Science and Engineering

Series Aalto University publication series DOCTORAL DISSERTATIONS 49/2012

Field of research computer science, security

Manuscript submitted 23 January 2012 **Manuscript revised** 10 April 2012

Date of the defence 25 May 2012 **Language** English

☐ **Monograph** ☒ **Article dissertation (summary + original articles)**

Abstract

Traditional credential solutions have well-known drawbacks. Purely software-based credentials are vulnerable to many attacks, while hardware-based security tokens and smart cards are expensive to deploy and, due to their typical single-purpose nature, force users to carry multiple hardware credentials with them. Recently, general-purpose security elements and architectures have started to become widely available on many commodity devices. On mobile devices, ARM TrustZone is a widely adopted security architecture. Such trusted execution environments enable realization of credentials that combine the flexibility of software solutions with the higher level of protection traditionally offered only by hardware credentials.

In this dissertation, we present several aspects of On-board Credentials (ObC), a novel credential platform for mobile devices. The ObC platform allows flexible creation of arbitrary credentials that utilize hardware security mechanisms for higher level of security. We challenge the prevailing thinking that a credential system must be centralized and closed in order to provide a sufficient level of security and usability. The distinguishing feature of the ObC platform is an open provisioning model that allows any service provider to deploy new credential instances to end-user devices without having to request approval from a centralized authority.

We study credential life-cycle management in open credential systems and present novel protocols for credential migration, temporary disabling and updates. We also describe mechanisms for key and application attestation. Our application attestation model makes property-based attestation practical by bootstrapping application authentication from existing certification infrastructures. We also compare open and closed credential platforms and show that openness does not have to imply decreased security or usability.

We have implemented the On-board Credentials platform for Symbian phones using the TrustZone trusted execution environment. Our implementation is part of the latest Nokia Symbian devices, and the On-board Credentials platform is currently being ported to other smartphone platforms. The first substantial credential deployments are now starting.

Keywords security, credentials, trusted computing, mobile devices

ISBN (printed) 978-952-60-4597-9 **ISBN (pdf)** 978-952-60-4598-6

ISSN-L 1799-4934 **ISSN (printed)** 1799-4934 **ISSN (pdf)** 1799-4942

Location of publisher Espoo **Location of printing** Helsinki **Year** 2012

Pages 186 **The dissertation can be read at** http://lib.tkk.fi/Diss/

A! Aalto-yliopisto

Tiivistelmä

Aalto-yliopisto, PL 11000, 00076 Aalto www.aalto.fi

Tekijä
Kari Kostiainen

Väitöskirjan nimi
Avoin malli avainten ja salaisuuksien turvalliseen hallintaan mobiililaitteissa

Julkaisija Perustieteiden korkeakoulu

Yksikkö Tietotekniikan laitos

Sarja Aalto University publication series DOCTORAL DISSERTATIONS 49/2012

Tutkimusala tietotekniikka, turvallisuus

Käsikirjoituksen pvm 23.01.2012 **Korjatun käsikirjoituksen pvm** 10.04.2012

Väitöspäivä 25.05.2012 **Kieli** Englanti

☐ **Monografia** ☒ **Yhdistelmäväitöskirja (yhteenveto-osa + erillisartikkelit)**

Tiivistelmä

Perinteisissä avainten ja salaisuuksien tallentamiseen ja hallintaan liittyvissä tietoturvaratkaisuissa on tunnettuja ongelmia. Puhtaasti ohjelmistopohjaiset menetelmät ovat haavoittuvia monille hyökkäyksille, kun taas älykortteihin ja vastaaviin turvallisuuslaitteisiin perustuvien ratkaisujen käyttöönotto on palveluntarjoajille kallista ja käyttäjien kannalta usein hankalaa. Viime aikoina yleiskäyttöiset laitteistopohjaiset tietoturva-arkkitehtuurit ovat alkaneet yleistyä monissa kuluttajalaitteissa. Esimerkiksi useat mobiililaitteet tukevat ARM TrustZone -arkkitehtuuria. Tällaisten laitteistopohjaisten turvallisten suoritusympäristöjen avulla on mahdollista toteuttaa ratkaisuja, joissa yhdistyvät perinteisten ohjelmistomenetelmien helppokäyttöisyys ja laitteistoratkaisujen tarjoama korkeampi turvallisuuden taso.

Tässä väitöskirjassa esitämme useita ominaisuuksia kannettaville laitteille suunnitellusta uudenlaisesta alustasta nimeltä On-board Credentials. On-board Credentials -järjestelmän merkittävä piirre on avainten ja salaisuuksien avoin asennusmalli, jonka ansiosta palveluntarjoajat voivat vapaasti kehittää ja käyttöönottaa uudenlaisia tietoturvaratkaisuja monenlaisiin käyttökohteisiin. Tutkimme tällaisten kredentiaalien hallintaa ja esitämme ratkaisuja kredentiaalien siirtämiseen laitteiden välillä, väliaikaiseen poistamiseen ja päivittämiseen. Olemme kehittäneet uudenlaisia menetelmiä, joiden avulla ulkoiset tahot voivat varmistua On-board Credentials -järjestelmässä talletettujen avainten ja salaisuuksien turvallisuudesta ja tällaisia kredentiaaleja käyttävien sovellusten luotettavuudesta. Analysoimme avointa asennusmallia ja osoitamme, että avoimuus ei vähennä järjestelmän turvallisuutta.

Olemme toteuttaneet On-board Credentials -järjestelmän TrustZone-arkkitehtuuria tukeville mobiililaitteille. Meidän toteutuksemme on osa Nokian uusimpia Symbian-laitteita ja järjestelmän mukauttaminen uusille alustoille on parhaillaan käynnissä. On-board Credentials mahdollistaa uusien palveluiden turvallisen, helppokäyttöisen ja edullisen integroinnin mobiililaitteisiin. Ensimmäiset merkittävät palveluiden käyttöönotot ovat tällä hetkellä alkamassa.

Avainsanat tietoturvallisuus, avainten ja salaisuuksien hallinta, mobiililaitteet

ISBN (painettu) 978-952-60-4597-9 **ISBN (pdf)** 978-952-60-4598-6

ISSN-L 1799-4934 **ISSN (painettu)** 1799-4934 **ISSN (pdf)** 1799-4942

Julkaisupaikka Espoo **Painopaikka** Helsinki **Vuosi** 2012

Sivumäärä 186 **Luettavissa verkossa osoitteessa** http://lib.tkk.fi/Diss/

List of publications

This dissertation presents an extended summary of the following publications:

P1 Kari Kostiainen, Elena Reshetova, Jan-Erik Ekberg and N. Asokan. Old, New, Borrowed, Blue–A Perspective on the Evolution of Mobile Platform Security Architectures. In *Proceedings of the ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 13–24, February 2011.

P2 Kari Kostiainen, Jan-Erik Ekberg, N. Asokan, and Aarne Rantala. On-board Credentials with Open Provisioning. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 104–115, March 2009.

P3 Kari Kostiainen, N. Asokan and Alexandra Afanasieva. Towards User-Friendly Credential Transfer on Open Credential Platforms. In *Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS)*, pages 395–412, June 2011.

P4 Kari Kostiainen, Alexandra Dmitrienko, Jan-Erik Ekberg, Ahmad-Reza Sadeghi and N. Asokan. Key Attestation from Trusted Execution Environments. In *Proceedings of the International Conference on Trust and Trustworthy Computing (TRUST)*, pages 30–46, June 2010.

P5 Kari Kostiainen, N. Asokan and Jan-Erik Ekberg. Credential Disabling from Trusted Execution Environments. In *Proceedings of the Nordic Conference in Secure IT Systems (Nordsec)*, pages 171–186, October 2010.

P6 Kari Kostiainen, N. Asokan and Jan-Erik Ekberg. Practical Property-Based Attestation on Mobile Devices. In *Proceedings of the International Conference on Trust and Trustworthy Computing (TRUST)*, pages 78–92, June 2011.

P7 Kari Kostiainen and N. Asokan. Credential Life Cycle Management in Open Credential Platforms (Short Paper). In *Proceedings of the ACM workshop on Scalable Trusted Computing (STC)*, pages 65–70, October 2011.

Author's contribution

- Publication **P1** presents a survey of common hardware-based security mechanisms and four platform security architectures on mobile devices, and it identifies open problems in mobile device security. This particular publication was the result of a team effort. The author's contribution was in surveying two of the operating system security architectures. The survey of the other two platform security architectures and of the hardware-based security mechanisms were done by the other authors. The author also contributed to the description of open problems.
- Publication **P2** describes a novel credential platform architecture, isolation mechanisms and an open provisioning model for hardware-based trusted execution environments (TEEs). The design and the implementation of the credential platform was also the result of a team effort. The author contributed to the design of the overall system architecture, the open provisioning model, as well as the isolation mechanisms. The author was independently responsible for the more detailed design and implementation of the operating system level security architecture. The design and implementation of TEE-resident components was primarily done by the other authors.
- Publication **P3** presents a user-friendly credential transfer protocol for open credential platforms. This work is the contribution of the author.
- Publication **P4** describes a novel key attestation mechanism for credential platforms like On-board Credentials. This work is the contribution of the author.
- Publication **P5** addresses temporary disabling of credentials implemented on embedded TEEs. This work is the contribution of the author.
- Publication **P6** describes a practical property-based attestation scheme that bootstraps from existing mobile application certification infrastructures. This work is the contribution of the author.
- Publication **P7** presents a comparative analysis between credential lifecycle management in open and closed credential platforms. This work is the contribution of the author.

Other publications

The following publications are not included to this dissertation. Publications **P8** and **P9** are closely related and referred to in this dissertation.

P8 Sven Bugiel, Alexandra Dmitrienko, Kari Kostiainen, Ahmad-Reza Sadeghi and Marcel Winandy. TruWalletM: Secure Web Authentication on Mobile Platforms. In *Proceedings of the International Conference on Trusted Systems (INTRUST)*, pages 219–236, December 2010.

P9 Jan-Erik Ekberg, N. Asokan, Aarne Rantala and Kari Kostiainen. Scheduling execution of credentials in constrained secure environments. In *Proceedings of the ACM workshop on Scalable Trusted Computing (STC)*, pages 61–70, October 2008.

P10 Nitesh Saxena, Jan-Erik Ekberg, Kari Kostiainen and N. Asokan. Secure Device Pairing Based on a Visual Channel (Short paper). In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 306-313, May 2006.

P11 Nitesh Saxena, Jan-Erik Ekberg, Kari Kostiainen and N. Asokan. Secure Device Pairing Based on a Visual Channel: Design and Usability Study. *IEEE Transactions on Information Forensics and Security (TIFS)*. 6(1):28–38, March 2011.

P12 John Solis, N. Asokan, Kari Kostiainen, Philip Ginzboorg and Jörg Ott. Controlling Resource Hogs in Mobile Delay-Tolerant Networks. *Computer Communications*, 33(1):2–10, January 2010.

P13 Kari Kostiainen, Olli Rantapuska, Seamus Moloney, Virpi Roto, Ursula Holmström and Kristiina Karvonen. Usable Access Control Inside Home Networks. In *Proceedings of the IEEE International Workshop on Trust, Security, and Privacy for Ubiquitous Computing (TSPUC)*, pages 1–6, June 2007.

P14 N. Asokan, Kari Kostiainen, Philip Ginzboorg, Jörg Ott and Cheng Luo. Applicability of Identity-Based Cryptography for Disruption-Tolerant Networking. In *Proceedings of the ACM workshop on Mobile Opportunistic Networking (MobiOpp)*, pages 52–56, June 2007.

P15 N. Asokan, Seamus Moloney, Philip Ginzboorg and Kari Kostiainen. Visitor Access Management in Personal Wireless Networks. In *Proceedings of the IEEE International Symposium on Multimedia (ISM)*, pages 686–694, December 2005.

Acknowledgments

This dissertation is based on my work at On-board Credentials project in Nokia Research Center, Helsinki. It has been a long project and during that time I have had the pleasure of working with many people. There are two persons who I would like to thank in particular. First, I am more than grateful for my advisor and team leader N. Asokan for all the excellent guidance throughout these years. Second, I want to thank my colleague Jan-Erik Ekberg for all the joint work and support. It is fair to say that this dissertation would not have been possible without you two. It has been a privilege to work with such brilliant people. Thank you guys!

In addition, I would like to thank all the other colleagues that have contributed to this work. Aarne Rantala, Elena Reshetova, Pekka Laitinen, Sampo Sovio, Pasi Eronen, Sandeep Tamrakar, Silke Holtmanns, Jörg Brakensiek and Valtteri Niemi all had a part to play. I would also like to thank our external research partners. Anders Rundgren and Magnus Nyström provided useful input for the initial system design. Ahmad-Reza Sadeghi and Alexandra Dmitrienko contributed to key attestation and several other related topics. Alexandra Afanasieva helped me with credential transfer protocol modeling. Elena Reshetova and Sven Bugiel provided me useful feedback on drafts of this dissertation.

Finally, I would like to thank my supervisor Tuomas Aura for helping me to finalize this work and my pre-examiners Mohammad Mannan and Jonathan McCune for their thorough reviews.

Kari Kostiainen
April 10, 2012
Helsinki

Contents

Chapter 1

Introduction

1.1 Motivation

Traditional credential types have well-known drawbacks. User memorable passwords suffer from bad usability and are vulnerable to phishing [47] and dictionary attacks. Applications such as web browsers may provide better usability and phishing protection by storing passwords on behalf of the user, but such *software credentials* can be compromised by exploiting operating system security vulnerabilities and by launching physical attacks.

Smart cards and dedicated security tokens can provide a higher level of security, but such *hardware credentials* have their own shortcomings. First, most existing hardware credentials are single-purpose devices and thus users are forced to carry multiple tokens with them. Second, although the current smart card standards [74] and platforms [118] support multiple credentials on a single physical card, in practice all deployed smart card systems are closed environments in which the installation of new credentials requires permission from the smart card issuer. As a result, deployment of services that utilize hardware-level security has been limited to major services providers, such as mobile network operators, banks, public transport authorities, and large corporations, which either can issue their own smart cards, negotiate agreements with external smart cards issuers, or purchase security tokens from external vendors. Deployment of hardware-protected credentials has so far been economically infeasible for most small service providers.

The communication and user interaction models of mobile devices are constantly evolving. For example, Near Field Communication (NFC) is an emerging short-range wireless technology that enables novel touch-based user interaction models. Recently, there has been significant interest for integrating services that traditionally have required separate smart cards or security tokens such as payment and ticketing into NFC enabled smartphones. The benefits of such integration are obvious: the user does not have to carry separate smart cards or tokens with him, and the communication and user interaction capabilities of smartphones enable improved overall user experience. Consider, for example, a public transport ticketing credential integrated into a smartphone. The user may check his current ticket balance and purchase new tickets directly from his device—features that would not be possible with traditional ticketing solutions.

Recently, various industry bodies have proposed system architectures with a centralized credential issuer for next-generation credential systems on NFC smartphones [68, 77, 39]. In essence, these proposals continue the tradition of closed credential systems and neglect the significant source of innovation that could originate from smaller, independent service providers and developers if they were allowed to develop and deploy their own hardware-level security solutions freely. Google Wallet [78] is the first noteworthy credential deployment for NFC smartphones that follows such closed installation model and the rest of the industry seems to be moving towards the same direction.

1.2 Research topic

During the past decade general-purpose hardware-based *trusted execution environments* (TEEs) have started to become widely deployed in various commodity devices. On PC platforms, the Trusted Platform Module (TPM) [83] is a commonly supported security element while many current smartphones support the ARM TrustZone [9] security architecture that augments the central processing core of the mobile device with secure storage and isolated execution capabilities.

The work presented in this dissertation is part of On-board Credentials project at Nokia Research Center. The goal of this project is to develop a *credential platform* that is simultaneously open, secure, usable and inexpensive using such general-purpose trusted execution environments.

- **Open.** Any service provider or application developer should be able to develop and deploy new credential types and instances into end-user devices without having to ask for approval, which may be subject to a business agreement, from a third-party controlling authority. Credential installation should be only subject to user approval.

- **Secure.** The credentials stored and executed on the credential platform should be resistant to common software attacks and simple physical attacks. The credential platform should provide a level of security that is comparable to traditional hardware credentials.

- **Usable.** Installation, usage and management of such credentials should be convenient for the users and service providers alike.

- **Inexpensive.** The deployment of the credential platform should be cost efficient. Instead of requiring new hardware investments from the device manufacturers and service providers, the credential platform should be based on hardware features available on existing commodity devices.

The primary aim of *this dissertation* is to design and implement novel security mechanisms that contribute to realization of such a credential platform in practice. In this dissertation, we address the *entire life-cycle* of credentials including the initial credential provisioning and secure storage, the credential execution, and the subsequent credential management by both credential issuers and end-users. We challenge the prevailing thinking that a credential system must be centralized and closed in order to provide a sufficient level of security and usability for credential issuers and end-users, and we aim to

show that a more open credential system can be designed based on the existing devices and hardware-based TEEs. Our intention is to provide a complete solution that can be deployed using existing devices today.

Another, traditionally important property of a credential platform has been the ability to monetize credential installation. While we acknowledge that such business models may still be valuable to mobile device manufacturers and mobile platform providers in the short term, our intention here is to study open credential platforms that enable free innovation around various credential solutions and thus improve the competitiveness of the mobile platform in the long run.

1.3 Contributions

In this dissertation, we present several novel security mechanisms for building a practical credential platform for existing mobile devices. As the basis of our credential platform design we have chosen ARM TrustZone security architecture for mobile devices, and while some of our key design decisions are based on the specific features and restrictions of this particular trusted execution environment, we argue that most of the mechanisms described in this dissertation can be applied to other trusted execution environments as well.

More precisely, we make the following contributions:

- **Credential platform.** We present a practical design and an implementation for a novel credential platform called On-board Credentials. The key features of the platform are a lightweight isolation environment and an open provisioning model. The credential platform architecture, the isolation environment and the provisioning model have been originally presented in publication **P2**. The credential platform design and implementation has been the result of team work with significant contributions from the author.

- **Life-cycle management.** We also study credential life-cycle management in open credential systems. We have identified issues, such as the lack of non-volatile secure memory and trusted user interfaces, which make credential management challenging and designed solutions to address these issues. Our credential transfer mechanism was originally presented in **P3**, temporary credential disabling protocol in **P5**, and credential updates and revocation in **P7**.

- **Attestation.** Additionally, we address remote attestation of hardware-based credentials and applications. First, we describe a novel key attestation mechanism for hardware-based credential platforms (**P4**). Second, we present a property-based application attestation scheme that bootstraps security from application signing infrastructures and platform security mechanisms commonly available in most current mobile platforms (**P6**). To the best of our knowledge, this application attestation mechanism is the first practical realization of the well-known concept of property-based attestation.

- **Comparative analysis.** We present a comparative analysis between open and closed credential platforms and conclude that open platforms can provide comparable security and user experience, with certain assumptions (**P7**). We also compare our credential platform with similar systems, implemented using other trusted execution environments and identify use cases and application areas that are particularly well suited for the ObC system.

- **Deployment experiences.** Finally, we share our experiences in implementing and deploying On-board Credentials platform. Our implementation is included in the latest Nokia Symbian devices, implementations for other mobile platforms are on-going, and the first noteworthy credential deployments are currently starting. For example, the New York Metropolitan Transportation Authority (MTA) is piloting transport ticketing with ObC-enabled NFC phones [123].

These contributions meet our design goals in the following ways. Our credential platform supports *openness* with a provisioning model that allows any service provider to develop and deploy new credential types and instances without having to obtain a permission from a centralized trusted authority. Usage of TrustZone hardware *security* mechanisms enables realization of credentials that are resistant to many common attacks and provide security levels comparable to traditional hardware credentials. We have formally verified security properties of selected parts of our security mechanisms. We identify several *usability* challenges, especially related credential life-cycle management, and we address user interaction issues throughout our credential platform design. However, a thorough usability analysis with comprehensive user studies is out of scope of this dissertation and left for future work. Our credential platform is *cost-efficient* in the sense that it builds on hardware security features supported by existing devices. Our credential provisioning, execution and management mechanisms can be deployed to many commodity devices that provide only very limited resources.

We argue that this work is a significant step towards the vision of a personal trusted mobile device that can integrate many security-critical credentials from transport ticketing to payments, corporate network access, opening doors, and more.

1.4 Outline

The rest of this dissertation is organized as follows. In Chapter 2 we provide background information: we explain the drawbacks of traditional software and hardware credentials, give a brief introduction to smart card security, introduce trusted execution environments that are commonly available in PC platforms and mobile devices, and discuss widely supported operating system level security frameworks. In Chapter 3, we identify more precise requirements for our credential platform. Additionally, we describe our assumptions and define a precise attacker model.

Chapter 4 presents the On-board Credentials platform. We describe our system architecture, provisioning model, isolation mechanisms, credential life-cycle management protocols and attestation mechanisms. We also describe

our implementation briefly. In Chapter 5, we analyze our work: we summarize the security properties of the ObC platform, compare open and closed provisioning models, discuss related work, explain how the ObC platform can be adapted to other environments, discuss promising use cases, and identify directions for future research. We finish this dissertation with some concluding remarks in Chapter 6.

Chapter 2

Background

2.1 Software credentials

Passwords are currently the most widely used user authentication mechanism for on-line services. User memorable passwords suffer from bad usability and low security. Users pick passwords with low entropy and re-use the same password for many services [166, 1]. Passwords are also vulnerable to phishing [47]. Two well-known approaches for improving the usability of on-line user authentication exist. First, many web browsers and browser extensions store user passwords [135, 85, 97]. Second, novel web authentication mechanisms, such as BrowserID [103, 89], avoid the use of user memorable passwords and instead rely on different types of locally stored web authentication credentials. We call such approaches *software credentials*.

While software credentials may provide an inexpensive solution for increasing usability and, in the case of web authentication, prevent phishing, software credentials are also vulnerable to many attacks. If the operating system does not provide proper runtime isolation between applications, a malicious application can read secrets stored by a software credential. If the attacker manages to compromise the device operating system at runtime or install a modified version of the operating system to the device, he may recover secrets stored by a software credential. Finally, if the attacker has physical access to the device, he may modify and read secrets stored in the device permanent storage while the device is turned off.

Thus, software credentials do not, in general, provide a sufficient level of security in use cases in which (a) the underlying operating system does not provide reliable isolation between applications, (b) the attacker may have physical access to the device, (c) the user himself may have an incentive to circumvent the system, or (d) the operating system itself cannot be trusted due to its complexity, and thus extensive attack surface. Consider, for example, a public transport ticketing credential that maintains the ticket value locally on the device implemented as a software credential. A malicious user could increase his ticket balance simply by modifying the ticket value when the device is turned off or duplicate the ticket by copying the credential from one device to another.

2.2 Security tokens

Many banks and large corporations issue *security tokens* to their customers and employees for two-factor authentication. RSA SecurID [91] is an example of a widely adopted security token that provides a display for showing one-time passcodes. The passcodes are typically derived from a secret that is shared with the authentication system backend. The user is expected to read the passcode from the token and enter it into the application he wishes to use for authentication together with a PIN code. Other types of security tokens exist as well. For example, in the model outlined in [162], the security token acts as a proxy in the connection, and the security token input and output mechanisms are used to confirm security-critical transactions, such as on-line banking payments, requested by the untrusted end-user device.

While security tokens can offer increased security compared to software credentials, these solutions have their own shortcomings, too. First, security tokens are typically single-purpose devices. One token can only be used to access one system or to improve the attack-resistance of a single service, such as on-line banking. Users need a variety of credentials, and carrying a large number of tokens is not attractive. Second, security tokens are expensive to manufacture and deploy. Thus, the use of security tokens has been feasible only for large organizations. Third, integration of security tokens to services and applications has proven to be problematic in terms of both security and usability [129]. Reading a passcode from one device and typing it into another is inconvenient for the users and is vulnerable to attacks in which the attacker has managed to compromise the application on the target device.

2.3 Smart cards

Smart cards are currently the most widely adopted hardware-based trusted execution environment with application areas ranging from cellular authentication to payments, public transport ticketing and physical access control [118]. The first smart cards were highly-constrained devices with limited memory and processing power, and supported only a single application. Modern high-end smart cards are capable of running more sophisticated operating systems and application development platforms [133]. Smart card standards are specified by GlobalPlatform [74].

JavaCard platform. On modern smart cards, the most widely used application development platform is JavaCard [118], although other smart card platforms, such as MULTOS [110], exist as well. In the JavaCard security model, application installation is controlled by code signing. Each installed application has to be signed and the signatures are verified using pre-established security associations. Typically only the smart card issuer may sign applications, but in some cases installation rights may be delegated to selected external authorities, such as pre-approved application developers. In the JavaCard security model, each signing authority has its own *security domain*. JavaCard applications are executed within the JavaCard virtual machine sandbox. Access to smart card operating system resources is only allowed, in a controlled manner, through the virtual machine. By default, smart card applications are isolated from one another during execution and in terms of persistent stor-

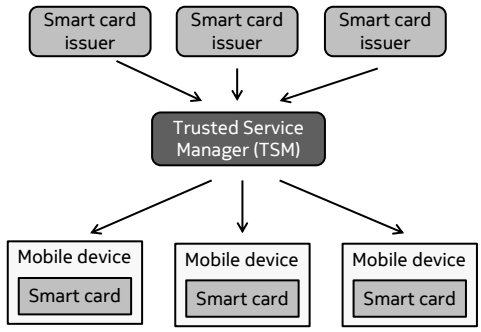

Figure 2.1: Trusted Service Manager (TSM) architecture

age, although controlled data sharing is permitted within a security domain [74]. The JavaCard security model is based on well-established security techniques; the virtual machine acts as a reference-monitor [4, 79] that provides controlled access to system resources, while principals (i.e., security domains) are assigned with code signatures [161].

The JavaCard sandboxing model relies on the correctness of the application bytecode. Malformatted bytecode that, for example, makes invalid object references or type casts or calls directly private methods may cause unwanted behavior [107]. Thus, JavaCard bytecode must be verified before execution. The first smart cards did not support on-card bytecode verification due to lack of resources, and many smart card issuers deployed pre-verified and signed applications. Since then, it has been shown that on-card bytecode verification is feasible even with very limited resources [46, 108] and the current high-end smart cards support on-card bytecode verification [118].

Smart card security has been studied extensively over the years. Security properties of smart card platforms have been formally modeled and verified [14, 6, 87, 7, 142]. Many smart card manufactures follow the Common Criteria security certification guidelines [41] during the design, manufacturing, personalization, testing and documentation phases, and thus many smart cards have been granted high security certification levels. For example, Caernarvon is a high-assurance smart card operating system designed for the highest possible security certification level [156, 93]. Many smart cards also provide protective measures, such as randomized circuit design, to complicate physical attacks [5].

Trusted Service Manager. The expected wide deployment of NFC technology in smartphones has caused high expectations for the integration of payment, ticketing, loyalty card and physical access control credentials into mobile devices [134]. To facilitate such deployment, several industry bodies have recently proposed credentials system architectures based on smart cards in which credential installation and management are controlled by a centralized authority called the Trusted Service Manager (TSM) [68, 77, 39]. Figure 2.1 illustrates such an architecture.

In these proposals, the TSM serves two purposes: First, it attempts to provide a convenient and centralized credential installation, update and backup point for users—analogous to current smartphone application stores. Second,

the TSM acts as a trusted intermediary between credential providers and smart card issuers. Since each smart card application installation requires acceptance from the smart card issuer, a large number of agreements would be needed in a heterogeneous environment with many credential providers and smart card issuers. With the TSM as a trusted intermediary, the smart card issuers could authorize the TSM with the application signing capability, and each credential provider could negotiate credential installation rights with one central authority. The recently launched Google Wallet [78] is the first major deployment of a credential system for NFC enabled smartphones following these principles.

Limitations. Smart card systems are closed environments; application installation always requires a permission from the smart card issuer. New approaches, like the recently proposed TSM architectures [68, 77, 39], may lower the barrier for obtaining application installation rights but, in essence, smart card systems remain closed. Since deployment of their own smart cards is infeasible for most small service providers, credentials implemented on smart cards remains a viable choice only for major service providers.

Additionally, smart cards have usability and security issues. Smart cards that are tightly integrated to end-user devices and services, such as the SIM (Subscriber Identity Module) cards used in cellular authentication, have proven to be a convenient method to access services securely, while usage models in which the smart card needs to be inserted into a separate reader device have been found troublesome by many users [129]. Besides, smart cards lack user input mechanisms; PIN codes and other security-sensitive user input have to be handled via another, potentially insecure, device. Vulnerabilities in modern smart card systems are typically found in external communication and user input handling [51, 119].

2.4 Trusted Platform Module

Integrated hardware-based trusted execution environments have recently gained wide deployment in many commodity devices including PCs and mobile phones. On PC platforms, Trusted Platform Module (TPM) [83], defined by the Trusted Computing Group, is the most widely deployed hardware-based security element. TPM is a separate security chip that is typically integrated to the mainboard of the PC. TPM supports secure storage and isolated execution of predefined cryptographic operations. Each TPM is equipped with a statistically unique key pair which may be certified by the TPM manufacturer. Applications may create additional keys for their own use. The private parts of these keys never leave the TPM in plaintext format. TPMs have a number of registers. A typical use of these registers is to store measurements of each loaded software component when the system is booted [138]. Access to TPM keys may be bound to register values, which can be used to ensure that only the same application that created the key can access it, and register values may be signed with the certified key to prove the current system configuration to an external verifier. Additionally, TPMs provide small amounts of non-volatile secure memory and support secure monotonic counters, and thus TPMs can maintain state across device re-boots.

Execution within a TPM is limited to predefined cryptographic operations, such as signatures and decryption. When used in combination with a suitable processor—one supporting Intel's TXT [35] or AMD's SVM [90] extension—a TPM can provide an isolated and measured execution environment for arbitrary code. This mechanism is called *late launch*. In a late launch system, a dedicated processor instruction re-initializes the processor, disables direct memory access and interrupts and measures the code that should be executed in isolation. The measurement is extended to a TPM register in a way that is distinguishable from other measurements for possible reporting to an external verifier. Late launch provides protection against software attacks and compromise of the device operating system. Depending on the processor architecture and the size of the executed code, the late launched code is executed either from the processor cache or from the system main memory. In cases where main memory is used, the late launch based system is vulnerable to hardware attacks against the main memory [84] or attacks against the system bus communication between the processor and the main memory.

The Trusted Computing Group has also specified a standard called Mobile Trusted Module (MTM) [81]. Compared to TPM, the MTM standard does not specify an actual hardware chip but rather an interface that the mobile device manufacturers are allowed to implement in different ways. Preferably, the MTM implementation should be based on hardware security features. In most parts, the MTM interface is identical to the TPM interface, although a few mobile device specific additions have been made (e.g., to enable remote and user based ownership models).

2.5 TrustZone

On mobile devices, ARM TrustZone [9] is currently the most widely adopted hardware security architecture, although other similar security architectures such as TI M-Shield [10] exist as well. In contrast to a TPM, TrustZone is not a separate security chip. Instead, TrustZone allows mobile device designers to augment the hardware configuration of the device with the needed security features. In mobile devices several functionalities or hardware components are typically included in a single chip. Figure 2.2 illustrates a typical mobile device hardware configuration: the device main processor, cellular modem, small amounts of read-only memory (ROM) and random access memory (RAM), display and keyboard and interrupt controllers, and debug and trace ports are included in the main processing core of the device. These *on-chip* components are connected with an internal bus. The rest of the mobile device components, such as the system main runtime memory, flash memory elements, the display, a possible keyboard and antennas are typically implemented as external components. These *off-chip* hardware elements are connected with an external device bus.

Execution modes. In the TrustZone architecture, the device main processor execution can be divided into *secure mode* and *normal mode*. The main processor switches between these modes in a time-slicing manner. The secure mode is intended for execution of small pieces of security-critical code while the normal mode is intended for running the device operating system and applications. The designer of a mobile device hardware configuration defines

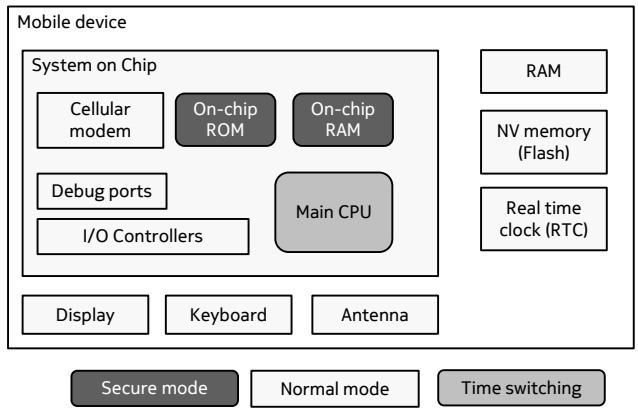

Figure 2.2: Typical mobile device hardware layout with TrustZone security architecture

the hardware components that are accessible in these two modes. In a typical configuration, access to on-chip memory elements is restricted to the secure mode only, while the device main memory and user input and output interfaces are accessible in the normal mode. In principle, the TrustZone architecture allows parts of the device main memory and input and output interfaces to be included to the secure mode but, in practice, current commodity mobile devices do not support this. The access control between different hardware elements is implemented by adding specific control signals to the system bus communication. The switch from secure mode to normal mode is only possible via a special *monitor mode* that provides a controlled transition from the less privileged mode to the more privileged mode. In a typical TrustZone deployment, all code executed within the secure mode has full access to all hardware elements configured for this mode. Thus, the trustworthiness of the code executed within the TrustZone secure mode must be verified either by code signing or by allowing only indirect and controlled access to secure mode resources.

Security features. In a typical mobile device configuration, the on-chip ROM is initialized with a device-specific key during the device manufacturing process. This key may be certified for external verification. Additionally, the on-chip ROM is typically configured with a trust root (e.g., a public key of the device manufacturer) and implementations of cryptographic algorithms (e.g., for verification of externally signed data). One of the most common uses of the TrustZone security architecture is *secure boot*. In the secure boot process, the device main processor is configured to boot from a specified location in the on-chip ROM. The code in this location, together with the trust root and the cryptographic implementation, is used to verify a signature over the first part of the system boot process (e.g., boot loader) within the secure mode, which in turn verifies the next part of the boot process (e.g., OS kernel). Such a chained *verify-before-execute* process can be used to verify the integrity of the software components that are executed during the device boot. By default, the ARM TrustZone security architecture supports integrity verification of

software components that are executed within the secure mode, but some mobile device manufacturers extend this functionality for integrity verification of the operating system code that is booted in the normal mode. Other common use cases of TrustZone include immutable device identifiers assigned during the device manufacturing, and the implementation of subsidy lock and digital rights management functionalities. For a more detailed discussion of hardware security on mobile devices, see **P1**.

The TrustZone architecture is based on well-established security mechanisms. In essence, TrustZone is a realization of *protection rings*, or protection domains, used in computer architectures since 1970s [144, 139]. In a typical ring architecture, ring zero is reserved for the operating system kernel while application code is executed on a higher and less privileged ring. Security-critical instructions, like direct access to hardware resources, are only possible in lower rings, and the transition to a lower ring always happens in a controlled manner. The TrustZone secure mode could be considered as ring -1; a protection domain that is more privileged than the operating system. Also, the basic principles of the TrustZone secure boot process had been outlined already in the 1990s [8].

Limitations and advantages. The design, manufacturing and testing processes of mobile devices typically do not follow commonly accepted security certification requirements. Thus, TrustZone does not offer similar certification levels as smart cards, for example. Compared to smart cards, a TrustZone based trusted execution environment (TEE) is also less resistant to hardware attacks since preventive mechanisms like randomized circuit design are not typically used in mobile devices. Additionally, the amount of memory available in the TrustZone secure mode is usually very limited.

The advantages of TrustZone are (a) higher processing speeds due to the use of the mobile device main CPU compared to the low-speed processors used in smart cards, (b) the possibility to tightly integrate the TEE into the end-user device and services and include user input and output handling hardware components as parts of the TEE, (c) the possibility to increase the TEE memory area by including parts of off-chip main memory to the secure mode for implementation of more complicated security services, and (d) the fact that no extraneous hardware is required, which enables realization of cost-efficient solutions for existing mobile devices.

2.6 Platform security

The current widely used PC operating systems are based on open application installation and discretionary access control typically implemented with user identities. Such security model has led to various problems, and the limitations of PC operating system security have been known for long—and even formalized [86]. Many improvements, such as operating systems based on fine-grained mandatory access control have been proposed (e.g., SELinux [111]). However, such models are complicated and laborious to configure and thus error-prone [92], and in practice the deployment of such systems has been limited. Currently, PC operating system security is mainly based on perimeter security solutions like firewalls and reactive security mechanisms such as anti-virus software.

Unlike PCs, mobile phones started as closed systems and application installation mechanisms were introduced gradually later. With the early mobile phones, end-users had come to expect a certain level of reliability from their devices. To retain that level of user trust while opening mobile phone platforms for application installation and Internet connectivity, the mobile device manufacturers introduced the necessary platform security models early in mobile phone development. Today, all prominent smartphone platforms support a platform security model. These security architectures are based on three basic principles:

Application signing and identification. In all current mobile platforms, applications must be signed before installation. In most platforms (e.g., iOS and Windows Phone), the application signing is done by a centralized trusted authority that runs the mobile application store. Other platforms (e.g., Android) support application signing by developers. In centralized architectures, mobile applications can be verified for security properties before publishing at the application store and installation can be limited to pre-approved applications, while in platforms that support developer-signing, the signature identifies the application developer. Most mobile platform security models also support runtime application identification. When an application makes an inter-process call towards another application or a system component, the called component may verify the identity of the caller. The application identities may be globally unique and centrally assigned by the application signing authority, or locally unique and linkable to developer identities.

Permission-based access control. In current mobile devices, access to system resources is controlled with *permissions*. In platforms with centralized application signing, the application permissions are assigned by a trusted authority, i.e., the mobile application store. In platforms that support developer-signing, the user is typically requested to approve the permissions requested by the developer at the time of application installation. At runtime, the mobile device platform security architecture acts as a *reference monitor* and provides controlled access to the system resources and services offered by other applications based on the assigned application permissions. On some mobile platforms with third-party application development based on managed code (e.g., Windows Phone), the reference monitor can be implemented as a part of the virtual machine that runs the applications, while other platforms implement the reference monitor as a part of the operating system middleware (e.g., Android). On platforms with application development based on native code (e.g., iOS and Symbian), the access control enforcement is typically done by a security subsystem of the operating system kernel. The granularity of the permission set varies substantially between mobile platforms. In some platforms (e.g., Android) the permission set is fine-grained and the application developers may define their own permissions while in others (e.g., Symbian and Windows Phone) the permission set is coarse-grained and fixed.

Application isolation. In all of these platforms, applications are isolated from one another, both at runtime and in terms of permanent data

storage. Each application typically has a private directory in the mobile device file system that other applications cannot access.

These recent mobile platform security architectures have borrowed substantially from older systems. The concept of a reference monitor has been defined and implemented in various systems starting from the 1970s [4, 79]. The first permission, or privilege, based operating system is the VAX/VMS system from the 1970s [88]. In the VMS security architecture, permissions are assigned to users, while in the current mobile platforms permission assignment is for applications. These recent mobile platform security architectures also resemble capability systems designed decades ago [104, 44, 163]. In fact, in some mobile platforms, the permissions are—somewhat misleadingly—called capabilities even though the assigned permissions cannot be moved from one application to another like capabilities could be. Principal assignment with code signatures, the application identification mechanism used in all of these platforms, was proposed first in the 1990s [161]. For more discussion on mobile platform security architectures, see publication **P1**.

These smartphone platform security models have proven to be effective in practice. Third-party application development for smartphones has become popular and users are accustomed to careless application installation. Yet, the damages caused by malicious smartphone applications have been limited—at least compared to PC platforms. Mobile malware that, for example, violates end-user privacy certainly exists, especially in mobile platforms where permission assignment is left to the user [63, 60]. However, the application isolation mechanisms cannot be easily circumvented which effectively prevents malicious applications from interfering with other applications, assuming that the attacker cannot compromise the platform security enforcement mechanisms themselves.

Chapter 3

Requirements

3.1 Assumptions

We have selected ARM TrustZone [9] as our primary TEE and the basis for our credential platform since it is the most widely supported hardware-based security architecture in current commodity smartphones and similar mobile devices. TrustZone enables various hardware and software configurations. We assume the following typical setup:

- *Certified device keys.* Each device is equipped with two statistically unique device keys: a symmetric device key and an asymmetric device key pair. (In practice, only a single symmetric secret may be configured to the device during manufacturing and the asymmetric device key pair may be derived from the symmetric device key using a suitable key generation algorithm during first device boot, for example.) We also assume that each device has a unique device identifier. The device keys and the identifier are configured to the on-chip ROM during the device manufacturing, and the on-chip ROM memory element is configured for TrustZone secure mode access only. The device manufacturer also issues a device certificate that binds the device identifier to the public part of the asymmetric device key.

- *Limited secure memory.* We assume that the mobile devices provide small amounts of on-chip RAM. Since the current commodity mobile devices have very limited on-chip secure RAM, we assume that only *a few tens of kilobytes* of on-chip RAM are available. The on-chip RAM is configured for secure mode access only. We assume that no external memory is included in the secure mode since this is not supported by current commodity mobile devices, and such an assumption provides us a stronger security model with respect to physical attacks against the device main memory [84]. We call the memory areas accessible only within the TrustZone secure mode *TEE resources*.

- *Secure boot.* We assume that the device manufacturer installs a trust root, typically the public part of the device manufacturer key pair, on the on-chip ROM during manufacturing. Implementations of common cryptographic routines (e.g., RSA and AES libraries) are configured to

the on-chip ROM during manufacturing. Using the fixed trust root and these cryptographic routines, the mobile device supports a hardware-based secure boot process (both for the software executed in TrustZone secure mode and the device operating system booted in normal mode).

- *No trusted user interface.* A direct and secure communication path between the user and the TEE is often denoted *trusted user interface* (see, e.g., [67] for more discussion). Within the very limited amount of on-chip RAM that mobile devices typically support, implementation of keyboard and display drivers is challenging if not infeasible. Thus, we assume that user input and output components are not included in the TrustZone secure mode and, thus, user input and output mechanisms are not part of our TEE.

- *No counters.* Neither do we assume on-chip non-volatile memory that could be used to implement monotonic secure counters within the TEE. Typical mobile devices lack secure non-volatile memory due to manufacturing cost reasons (see, e.g., [141, 37] for reasoning).

In addition to TrustZone TEE, we assume that the mobile device operating system supports a typical operating system level platform security architecture with (centralized) application signing, permission-based access control and application isolation. We assume that the integrity of the platform security architecture can be verified with a TrustZone based secure boot and access to the TEE can be limited to privileged system components with permissions.

Finally, we assume that no malicious code is inserted in the on-chip secure memory during the manufacturing of the mobile device, the controlled transition from the TrustZone normal mode to the secure mode cannot be circumvented, the device hardware is free of defects, the typical security properties of common cryptographic primitives will hold, and that the cryptographic libraries used in the TrustZone secure mode are free of implementation errors.

3.2 System model

Our purpose is to design an open credential platform that allows development of *arbitrary* credential types. Many security services can be implemented with pre-defined credential types. For example, RSA signatures and encryption and message authentication with symmetric keys are sufficient for several security services. However, many use cases require custom security algorithms, or credentials, that combine commonly used cryptographic primitives in an application-specific way. For example, SecurID [91] is a proprietary authentication algorithm used in physical authentication tokens, many on-line bank user authentication mechanisms rely on custom security algorithms, and most contactless public transport ticketing solutions apply their own authentication algorithms due to limited resources available. We aim to design a credential platform that allows development of any credential type—that is possible to implement within the hardware limitations of the existing TEEs, of course.

The credential issuers should be allowed to write code that implements their own security algorithms and will be executed within the TEE in a secure manner. We call such pieces of code *credential programs*. Respectively,

we call the confidential data (e.g., keys and passwords) that these programs operate on *credential secrets*. We assume that some credential programs may need to operate on multiple secrets at the same time and that in some cases several credential programs may be need to be combined in order to implement the desired security functionality. We call a combination of credential programs and secrets that are used to implement an intended service a *credential instance*.

We assume that credential developers and service providers have no prior trust relationship with the mobile platform provider or one another. (In many cases, the service providers do need a prior context with the end-users for ensuring that the credentials are provisioned to the correct individual.) In some cases, the credential programs and secrets may originate from different sources; an authentication system provider could, for example, write the credential program while the credential secret is issued by a different service provider. In general, we call providers of credential secrets and programs *credential issuers*.

3.3 Requirements

In this section, we identify more precise requirements for our credential platform. The high-level design goals are openness, security, usability for end-users and cost-efficiency. The credential platform should address the needs of all involved parties including service providers, credential developers and end-user. Finally, the credential platform should provide the functionality to support the entire life-cycle of a credential starting from the initial provisioning to secure storage and execution and subsequent management operations. Ignoring any of these aspects, is likely to prevent real-world adoption of our system.

Provisioning. Our first set of requirements is related to credential installation, or provisioning. In an open provisioning model, any service provider should be able to develop its own credential programs and deploy them securely with the associated credential secrets to end-user devices. More precisely, we identify the following requirements:

- *TEE authentication.* Credential secrets are confidential pieces of data, and this confidentiality should be preserved during credential provisioning. The credential issuer should be able to verify that the credential secrets are provisioned only to a compliant TEE that does not reveal the secrets.

- *Provisioning user authentication.* The credential issuer should be able to verify that the device with the compliant TEE belongs to the intended user. This ensures that only the correct user can use the provisioned secret. In practice this can be done with a provisioning password, for example. We explicitly do not require *credential issuer authentication*. In an open provisioning model, credential deployment should not be limited to known, or pre-approved, issuers.[1]

[1] In a password-based provisioning user authentication protocol, the credential issuer may be authenticated for password phishing protection. However, an open provisioning model,

- *Program authorization.* Since credential secrets and programs may originate from different issuers and mutual trust may not exist between these issuers, the credential secret issuer should be able to control that only credential programs authorized by it are allowed to access the credential secrets provisioned by it.

- *Authorized sharing.* Some credential programs may need to share data. This data may be credential secrets or other data produced by credential programs. Issuers of credential programs and secrets should be able to authorize credential programs that can share data between each other. Access to secret data by unauthorized credential programs should be disallowed.

- *Program confidentiality.* While most security algorithms are—and should be—public, in some cases also the credential program may be confidential. The widely adopted RSA SecurID authentication mechanism [91] is an example of a non-public security algorithm. Thus, the credential platform should ensure credential program confidentiality both during credential provisioning and during credential storage and execution.

Execution. After the provisioning, the credential platform should guarantee the security of the installed credentials during storage and execution. Obviously, the credential platform should prevent operating system level applications from revealing credential secrets. More specifically, we identify the following TEE-internal credential execution requirements:

- *TEE resource isolation.* Since credential programs are written by unknown (i.e., untrusted) developers, the credential platform should only allow credential programs to access the TEE resources such as device keys indirectly and in a controlled manner. Allowing direct access to the TEE resources would allow malicious credential programs to compromise other services implemented using these resources.

- *OS isolation.* Likewise, credential program execution should be isolated from any OS-level application and the OS itself. For example, an untrusted credential program should not be able modify data structures reserved for the operating system internal use or access data that an OS-level application has stored on the device persistent storage.

- *Program isolation.* Credential programs should be isolated from one another. The credential program isolation should hold both at runtime and for persistent data that is generated by credential programs and stored outside the TEE. Only explicitly authorized data sharing should be permitted.

- *Computation limitation.* The credential platform should limit credential program execution in terms of processing time.

as such, does not require issuer authentication for ensuring the security of the provisioned secrets. Only later credential authorization and management operations by the issuer need to be authenticated.

- *Replay protection.* The credential platform should have a mechanism for preventing the execution of old versions of credentials. Examples where this is needed include credentials that maintain state locally (e.g., a ticketing credential that uses a stored ticket value) and credentials that can be updated or revoked remotely. Because secure non-volatile memory or counters are not available in the devices we are addressing, alternative mechanisms to typical counter-based solutions are needed.

Besides controlling TEE-internal credential execution, the credential platform should also control the access to the credentials from outside the TEE:

- *Runtime user authentication.* The credential platform should be able to verify that the device is in possession of the correct user (i.e., the user for whom the credential was provisioned) when the credential is invoked. A typical example of a local user authentication mechanisms is to require a PIN code from the user before credential execution.

- *Application authentication.* In addition, credential platform should be able to control which OS-level applications are authorized to invoke TEE-protected credentials, for example, to prevent possible malware applications on the device from invoking credentials installed by and reserved for the use of other applications. Any application authentication mechanism necessarily relies on the integrity of the OS platform security architecture that provides application identification at device runtime.

Life-cycle management. Both credential issuers and end-users should be able to manage their credentials after the initial provisioning and installation. We identify the following credential management requirements:

- *Authorized updates.* Credential issuers might want to issue updates to the already provisioned and installed credentials (e.g., update a credential expiry date or a similar parameter). Each credential issuer should be able to update only the credentials provisioned by it. Unauthorized entities should not be able to update credentials.

- *Controlled migration.* Users switch mobile devices frequently and they should be able to migrate credentials from one device to another easily. At the same time, credential issuers should be able to control credential migration (e.g., the identity of the devices to which the previously provisioned credentials can be migrated) and define that certain credentials cannot be migrated and require explicit re-provisioning.

- *Temporary disabling.* Users should be able to temporarily disable credentials from and restore credentials to the device. Example use cases include lending the device to a friend or traveling to an unsafe location.

- *Secure backups.* Backups that can only be restored to the same device are trivial to implement by encrypting the credentials a device-specific key that is accessible only within the TEE of the device. However, the attacker (e.g., the user himself) should not be able restore outdated backups in case of credentials that maintain state.

Attestation. While many credential programs operate on remotely provisioned credential secrets (i.e., symmetric keys), the credential platform should also allow developers to create credential programs that generate new keys (i.e., asymmetric keys) locally on the device and then enroll these to services. This brings us to another requirement:

- *Key attestation.* When enrolling credential program generated asymmetric keys to externally provided services, the service providers should be able to (a) verify, or *attest*, that the enrolled key belongs to a compliant TEE and (b) identify the entities that are allowed to access or use that key on the device.

Besides verifying properties of TEE-protected credentials, in some cases a remote entity needs to verify the trustworthiness of the OS-level application that accesses credentials and services provided by the remote entity:

- *Application attestation.* A remote verifier should be able to determine which or what kind of an application is accessing its services. The application attributes that the service provider would need to know depend on the application scenario. Thus, the application attestation mechanism should be generic in terms of the attested attributes.

3.4 Attacker model

We address multiple attacker types. We assume (a) *remote attackers* who try to attack the system over the network, (b) *device owners* who have an incentive to duplicate or modify credentials stored on their own devices, and (c) *physical attackers* who may gain physical possession of a target device temporarily. For more detailed discussion, we define the following attacker capabilities.

Application installation. An attacker with this capability can install malicious applications to the target device. Remote application installation attacks are possible using well-known social engineering techniques, while physical attackers and device owners may install malicious applications through normal application installation mechanisms.

We acknowledge that in some smartphone platforms (e.g., Windows Phone and iOS), application installation always takes place through a centralized application store and only pre-approved applications can be installed to the device. However, malicious applications can be found also in smartphone platforms with centralized application vetting [63, 59]. Additionally, all smartphone platforms support developer modes in which unchecked applications can be installed to a limited number of devices.

Network control. An attacker with network control capability can read and modify all traffic between the target mobile device and any external device as assumed in the typical Dolev-Yao model [49]. The network control capability can be easily gained when the device is used in a public and unencrypted wireless network, for example.

OS compromise. Our third attacker capability means that the attacker is able to circumvent the security features provided by the underlying operating system such as the application isolation mechanisms and the permission-based access control. This attacker capability can be justified by the increasing size and complexity of current smartphone platforms. It is a widely acknowledged fact that the number of software bugs and vulnerabilities increases as the software system gets larger and more complicated (see, e.g., [101] for a summary of studies on the subject). As smartphone operating systems approach their PC counterparts in size and complexity, vulnerabilities become inevitable and, indeed, several exploits have been reported especially for smartphone platforms with an application development model based on native code [45]. Additionally, techniques like address space layout randomization (ASLR) that attempt to prevent common runtime attacks such as return oriented programming [148] have become mainstream in PC platforms but have not yet gained wide deployment in mobile devices primarily due to performance reasons [23].

On the other hand, one can argue that modern smartphone platforms can provide reliable protection against OS compromise—at least, compared to PC operating systems. Three arguments exist: (1) Some mobile device manufacturers (e.g., Nokia) deploy a hardware-assisted secure boot that ensures the integrity of the operating system after the device boot. (2) Some mobile platforms (e.g., Windows Phone) allow application development only using managed code with strong typing, bounds checking and memory management enforcements, which limit the chances of traditional runtime vulnerabilities, such as buffer overflow attacks. (3) Some mobile platforms (e.g., iOS) support advanced techniques like runtime code signing enforcement that make code injection attacks more difficult [167]. Thus, we argue that in some use cases it is a reasonable assumption to rely on the platform security enforcements provided by the operating system.

Simple off-line attacks. Attackers with physical possession of the device may read and modify data from persistent non-secure memory (e.g., off-chip flash memory) when the device is turned off, update the device firmware by common firmware update mechanisms, and replace off-chip memory elements on the device.

Simple runtime hardware attacks. Physical attackers with this capability may read and modify off-chip device bus communication at system runtime. Such runtime hardware attacks can be mounted with cheap, of-the-shelf hardware by attackers with a moderate skill set. Monitoring the off-chip device bus is possible, for example, through the JTAG interface with commonly available debugging tools such as bus protocol analyzers.

Sophisticated hardware attacks. Attackers with more sophisticated equipment and better knowledge may mount off-line and runtime attacks against the on-chip secrets and processing within the TrustZone secure mode. The sophisticated attacks include drilling small holes in the target chip with a laser and installing small probes with electronic

contacts to the integrated circuits [5], and non-invasive side-channel attacks such as differential power analysis [99] and differential electromagnetic analysis [132]. These attacks require higher skills from the attacker and relatively sophisticated tools.

In the rest of the dissertation, we assume that the attacker always has application installation, network control and simple off-line attack and simple runtime hardware attack capabilities. We exclude sophisticated hardware attackers from our threat model. We assume that the hardware-assisted secure boot holds but consider runtime OS compromise possible, and we discuss the implications of such compromise when our solutions rely on the integrity of the platform security architecture.

Chapter 4

On-board Credentials

4.1 Architecture

We start our credential platform design description by explaining the On-board Credentials architecture, which is illustrated in Figure 4.1. Our architecture includes four trusted components within the TEE. The trustworthiness of these components is verified by code signing, i.e., verifying signatures over them with respect to the device manufacturer trust root that is installed to the TEE during manufacturing.

- The *Provisioning* module converts remotely provisioned credential programs and secrets into locally encrypted, or *sealed*, structures that can be processed by the Interpreter. The credential provisioning and the local installation are intentionally separated from the credential execution to keep the implementation size of the Interpreter minimal. We will explain our provisioning model in Section 4.2.

- The *Interpreter* executes credential programs within the resource-constrained TEE and isolates credential programs from the TEE resources and from one another. The credential programs are written in a custom programming language and compiled into a custom bytecode designed for the ObC architecture. We will give more details about the bytecode format and the Interpreter operation in Section 4.3.

- The *Management* module performs credential transformations needed for credential transfer, disabling and credential updates. These transformations are discussed in Sections 4.4, 4.5 and 4.6.

- The *Key Engine* provides asymmetric key operations. Keys can be created and used by credentials programs within the TEE and by operating system level applications from outside the TEE. We describe key attestation in Section 4.7.

On the operating system side, our architecture includes a trusted software component called *Credential Manager*. The purpose of this component is to provide controlled access to TEE-resident credentials for OS-level applications. Through the ObC API, applications can install remotely provisioned

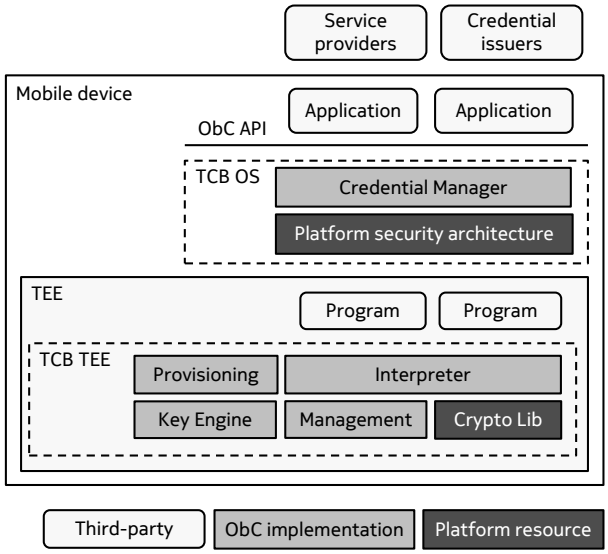

Figure 4.1: On-board Credentials architecture

credential programs and secrets, execute such credentials, create and operate with local asymmetric keys, and trigger life-cycle management operations such as credential disabling or migration. The Credential Manager also handles application attestation that is explained in Section 4.8.

The Credential Manager component performs access control for applications using runtime application identification and permission-based access control mechanisms that are provided by the operating system platform security framework. In addition, Credential Manager may handle local user authentication for credential execution. User authentication for credential provisioning is typically handled by OS level applications. Application identification and permissions are also used to deny direct TEE access from applications on the operating system kernel level. The Credential Manager component itself has permissions granted by the device manufacturer needed for TEE access. The TEE access enforcement relies on the integrity of the platform security architecture. Credential Manager also handles piecewise credential execution and hides the complexity of internal execution state management from the calling applications.

For the purposes of more detailed security analysis (see Section 5.1), we define *two* alternative definitions for the Trusted Computing Base (TCB) of the On-board Credentials system. First, assuming an attacker with OS compromise capability, the TCB of our system consists of the Interpreter, Provisioning, Key Engine and Management modules together with the implementations of the cryptographic libraries provided by the TEE. We call this set of trusted components *TCB TEE* (see Figure 4.1). Second, for operating system level access control and other similar security enforcement, we rely on the integrity of the platform security architecture (i.e., we assume that the attacker cannot circumvent runtime application identification and permission-based access control enforcement mechanisms), and thus for this alternative TCB model we include also the parts of the operating system that

implement these enforcements and the Credential Manager component in the set of trusted software. We call this extended selection *TCB OS*. For more discussion of the On-board Credentials architecture see publication **P2**.

4.2 Credential provisioning

Traditional credential provisioning models are based on pre-established security domains. In smart card provisioning, for example, the installed applications must be signed with respect to a trust root that is typically configured during manufacturing. The smart card issuer may authorize new security domains by issuing signed new trust roots, and data sharing within a security domain is possible. In an open provisioning model, such pre-established or post-authorized trust roots cannot exist since the identities of credential issuers are not known in advance and necessarily not even at the time of provisioning. Thus, our first research question can be stated:

Research question. How to establish the needed security domains *dynamically* for secure credential program provisioning and authorized data sharing? The implementation of such a provisioning model should work within the constraints of minimal TEEs on existing mobile devices.

The ObC provisioning model is based on security domains that are dynamically established based on a *same-origin policy*. A typical credential provisioning operation goes as follows. The credential issuer picks a new symmetric *family key*. Essentially, this key defines a new security domain. The family key is encrypted for the target device using the public part of the certified device key. Credential secrets and programs can be encrypted and authenticated using the family key. The credential issuer may authorize credential programs to access its secrets by authenticating with the family key a statement about the identity of the intended credential program. We call this process *endorsement*. The Provisioning module on the target device does not know the identity of the credential issuer but allows credential endorsements only if they are created with the family key that was used to provision the credential secret (i.e., same-origin policy).

The Provisioning module converts the provisioned credentials and endorsements to locally sealed data structures that the Interpreter can process without public key operations. The family key, or a local transformation of it, is included in these seals. This allows the Interpreter to determine the credential family at credential runtime and to create family-specific data seals that allow controlled data sharing between authorized credential programs. The separation of public key cryptography into a separate Provisioning module enables a minimal Interpreter implementation footprint and runtime memory requirements.

The TEE authentication and program confidentiality are based on device certificates. Provisioning user authentication must be handled by an operating system level software module, which relies on platform security. Our provisioning model has no freshness guarantee; an already provisioned credential can be re-installed to the same target device. If the target device supports secure counters, replay protection could be easily added to the provisioning protocol. Prior to us, the principle of the same-origin policy has been mainly

used in the context of web application isolation. For more detailed description of our provisioning model, see publication **P2**.

4.3 Credential isolation

The purpose of the Interpreter component is to isolate the credential programs from the TEE resources and from one another. Two well-known methods of ensuring that code received from an untrusted source does not violate the required security properties (e.g., isolation) exist. In a system based on *proof-carrying code* [122], the code issuer generates a formal proof of the security properties of the code, and the code receiver verifies this proof before executing the code. The primary problem with proof-carrying code is that code proofs are difficult to create. Assuming that every credential issuer would be capable of producing such formal proofs is not realistic. Additionally, such code proofs can be very large and processing them within the minimal TEE of existing devices would be challenging, if not infeasible.

The Interpreter in the ObC architecture relies on a more common isolation technique: sandboxing provided by an interpreted language. The smallest widely used bytecode interpreter is the JavaCard virtual machine. Typical JavaCard implementations with on-card bytecode verification require at least one hundred kilobytes of memory [118]. In our target TEE, the available secure memory is only a few tens of kilobytes. Thus, our next research question becomes:

Research question. How to design a bytecode set and a matching interpreter that can be implemented within the constraints of the minimal TEEs of existing devices? Besides being small in size and providing the required isolation properties, the bytecode set should be expressive enough for implementation of *arbitrary* credentials and allow authorized data sharing.

The ObC Interpreter is based on a small custom bytecode set. This set consists of about 40 instructions including stack manipulation, memory allocation, jump, and subroutine invocation. As a comparison, the JavaCard instruction set contains roughly 200 instructions. All data elements in the ObC bytecode are of the same type or arrays of that type.

An interpreted runtime environment typically has to provide three types of security enforcement. Control-flow safety and memory safety are easy to provide. The Interpreter simply ensures that the program code counter does not jump outside the program code memory area and that the array variables are not accessed outside their allocated buffers. Type safety, in general, is more difficult [130]. In the ObC architecture, the need for complicated type safety enforcement is eliminated with a bytecode set that supports only a single type. Implementation of type safety verification can require up to 5000 lines of code [128]. Therefore, with our single-type language, considerably smaller TCB can be achieved. Additionally, compared to smart cards, the ObC Interpreter provides a simpler isolation model. Whereas smart card applications have access to smart card resources such as a file system, the ObC credential programs have no direct access to the TEE resources. The

communication (i.e., the input and output parameter passing) between OS-side entities and credential programs is implemented with a shared memory buffer. The Interpreter ensures that credential programs do not read or write outside this memory area, which guarantees our OS isolation requirement.

The Interpreter operates in two phases. First, locally sealed credential programs are decrypted and program endorsements are verified with respect to the credential secret family. The second phase is a loop in which bytecode instructions are executed one by one. The Interpreter controls the execution time by limiting the number of execution steps. Only one program can be executed at a time.

The Interpreter provides interfaces to common cryptographic primitives. Credential programs can locally seal data they generate and also create new keys for the use of the same and other credential programs. For persistently stored data and keys, the access control is based on the provisioning families. Each credential program can seal data for persistent storage on the OS side with a key that is derived from the family key and the device-specific key with a one-way function.

Credential programs can be divided into subroutines. Subroutines facilitate both code reuse and implementation of larger credentials by splitting the credential logic into smaller pieces. For subroutine execution, the current program state (i.e., the execution stack) is sealed using a key that is unique to this particular credential execution to prevent the misuse of a saved credential state from another credential execution. Identities of both the calling main program and called subroutine are included in the intermediary sealed state to prevent unauthorized credential program switching. The Credentials Manager component maintains these intermediary seals and hides the complexity of piecewise execution from the calling application. For more details on the bytecode and the Interpreter see **P2** and [55]. Credential execution scheduling and intermediary sealing is discussed in more detail in **P9**.

4.4 Credential transfer

Migrating credentials from one device to another can be challenging in open credential platforms. While some credential issuers might allow their credentials to be directly copied from one compliant TEE to another, others might require explicit re-provisioning to the target device from the original credential issuer, for example, to control the number of credential instances the user has. In a closed and centralized credential system, such re-provisioning is straight-forward. The user simply authenticates to the centralized credential issuer from the target device, and all credentials may be copied to it in one go. In an open and decentralized credential platform, similar credential re-provisioning would require a separate user authentication for each credentials issuer—a user experience model that clearly would not be optimal for users that switch devices frequently.

Credential migration is also complicated by user interaction issues. Simply verifying that the credentials are transferred to a compliant TEE does not ensure that the credentials are migrated to the correct user. The migrated credentials should be securely bound to the identity of the user but, since current mobile devices typically do not support trusted user interaction

mechanisms, such binding must be handled on the operating system side. Our next research question is:

Research question. How to transfer or re-provision credentials from one device to another in a decentralized and open credential system assuming devices with no trusted user interface?

We have designed a server-assisted credential transfer that uses *delegated re-provisioning*. In this protocol, migratable credentials are copied to the target device directly, or via an assisting server, while non-migratable credentials can be automatically re-provisioned to the target device by the original credential issuers using a delegation mechanism that is mediated by the server.

We address the lack of secure user input by using the *trust on first use* principle. The user installs his identity to the device during first boot, when the device OS integrity can be trusted due to the hardware-assisted secure boot process. The credentials are bound to this identity during provisioning which prevents a later compromised system from migrating the credentials to another compliant device that belongs to a wrong user.

We have formally modeled and verified this credential transfer protocol (excluding user interaction) using the AVISPA tool [159]. Our transfer mechanisms is a practical solution that can be implemented on existing devices and integrated into typical mobile device initialization processes for good user experience. For more details, see **P3**.

4.5 Credential disabling

Temporary credential disabling from embedded TEEs is a seemingly simple problem. However, straight-forward solutions are not adequate to solve this issue. Password-based approaches are vulnerable to off-line brute force and dictionary attacks while simple solutions, in which the credential recovery is bound to a full-length key stored to a removable memory element, are both inconvenient and insecure. Considering users' careless handling of such removable elements, credential recovery keys are likely to eventually leak to the attacker. In a TEE that does not support non-volatile secure memory, leaked keys and matching backups cannot be easily invalidated. For more detailed problem description and for more thorough explanation on why straight-forward approaches fall short, we refer the reader to **P5**. In short, the temporary credential disabling problem can be formulated as follows:

Research question. How to prevent an attacker with temporary access to the target device from restoring previously generated credential backups on devices with no secure counters?

We have designed two solutions to address this problem. First, we outline a credential disabling solution based on the presence of a SIM (Subscriber Identity Module), which the user is expected to remove from the device when lending it. The unique identity of the SIM is bound to each credential backup, and the credential platform will restore credential backups only if the same SIM is present in the device. Secure implementation of such an approach requires a challenge-response protocol between the device TEE and the SIM

and, therefore, requires mobile network operator involvement. Second, we have designed and implemented a *server-based credential disabling mechanism*. In this approach, we mitigate the lack of non-volatile secure memory on the end-user device by using a server as a trusted state storage. However, we do not have to trust the server with the credentials themselves. More detailed descriptions of our temporary credential disabling solutions are presented in **P5**.

4.6 Credential updates

In an open credential platform, authorized credential updates can be achieved by utilizing the principle of same-origin policy that we apply in the provisioning. At the time of credential installation, the credential issuer defines a key (i.e., the family key), which is included in the provisioned credentials. The credential platform maintains the binding between this key and the locally sealed credentials. Later, authorized credential updates can be issued by signing them with the same key. We explain this credential update mechanism in **P7**.

4.7 Key attestation

In the On-board Credentials platform, credential programs as well as OS-level applications can create new asymmetric key pairs locally. When such keys are enrolled to remote services, the service provider should be able to verify that the key resides in a compliant TEE and to determine other useful attributes about the key such as identifying the entities that are allowed to use this key. Our next research question is:

Research question. Identify the exact key attributes that need attestation in a credential platform like On-board Credentials, and design a key attestation mechanism for those attributes.

We have designed a *two-level key attestation mechanism*. This attestation mechanism provides evidence on both TEE-internal access control attributes (*internal attestation*) and OS-level access control settings (*external attestation*) as well as typical key attributes such as key usage. The mechanism allows an external verifier to determine the credential programs that are allowed to access a key in terms of credential program and provisioning family identities. The OS-level applications that are allowed to invoke the attested key can be identified using application identifiers assigned by an application signing authority and maintained by the OS-level platform security architecture.

The internal attestation signatures are controlled by a trusted component within the TEE while external attestation signatures are controlled by the Credential Manager on the operating system side. We use *two distinct device keys* for the internal and external attestation signatures. As a result, the attestations of TEE-internal properties are not vulnerable to a possible runtime compromise of the device OS. Both keys are stored within the TEE for offline attack protection. If the device is issued only a single certified key

during manufacturing, the second device key may be bootstrapped from the first one by running an on-line protocol with the device manufacturer. For more discussion of key attestation, we refer the reader to publication **P4**.

4.8 Application attestation

In traditional *binary-attestation*, exact measurements of software components are sent to an external verifier to determine if the system is compliant. Such attestation has privacy issues and it is hard to realize in modern system with large software components and frequent updates. Attestation of more abstract properties has been proposed as a solution [136]. In *property-based attestation*, a trusted authority defines mappings from exact measurements to higher-level properties, and the attesting device reports the matching properties to the external verifier.

Although the concept of property-based attestation has been known for long, no practical deployments have emerged. One reason for this is that so far there has been no use case in which property-based attestation is both urgently needed and practical to deploy. In many cases, it is unclear what the high-level properties should be, and who would have an economic incentive for maintaining such a trusted translation service—a non-trivial task that requires maintenance due to frequent software updates [12].

Research question. How to make deployment of property-based attestation feasible for a concrete use case in the context of mobile devices?

We have designed a novel property-based attestation mechanism for mobile devices. We mitigate the problems of previous proposals by *bootstrapping from application certification infrastructures*. In our approach, we convert the centrally assigned application identities to properties that need attestation. Such translation requires few updates compared to traditional property-based attestation in which binary measurements are translated to higher-level properties. This eliminates, to a large extent, the need to introduce and maintain a new translation service. On the device, we utilize application identification mechanisms provided by the platform security architecture. For runtime protection, our solution relies on the integrity of the platform security architecture (TCB OS). The attestation key is kept within the TEE for off-line attack protection.

We have designed and implemented this attestation mechanism in the context of the MirrorLink system [24, 53]. MirrorLink is a widely supported industry standard for utilizing mobile device provided services such as navigation in cars. In MirrorLink, the primary security requirement is driving safety; the car head-unit must verify that the data it receives is trustworthy before presenting it to the driver. We have defined a *hierarchical attestation approach* in which content attestation is built on top of a property-based application attestation mechanisms that, in turn, relies on device authentication. Our attestation mechanism is part of the MirrorLink standard [53] and the first products supporting this attestation mechanism will soon emerge. The MirrorLink system is likely to be the first significant deployment of property-based attestation in practice. For the full details, see publication **P6**.

Component	Lines of code
Interpreter	3683
Provisioning	881
Key engine	1183
Management	688
Crypto Lib	$\sim$ 9700
TCB	$\sim$ 16000

Table 4.1: Component sizes in lines of code

4.9 Implementation

Our primary implementation target has been Nokia Symbian devices with the TrustZone TEE. Symbian devices supporting the M-Shield TEE were used for the first implementations (see, e.g., **P2**). The TEE-internal components have been implemented in the C language. The implementation sizes in terms of lines of code for these components (without headers or comments) are listed in Table 4.1. The size of the ObC Interpreter is less than 10 kB in a compiled format. As a comparison, a typical JavaCard virtual machine implementation requires more than 100 kB. Implementing a fully functional interpreter with such a minimal code footprint has required the use of various code optimization techniques.

For credential program developers, our implementation provides interfaces for AES-EAX [18] authenticated encryption and decryption, RSA-SHA1 signatures and their verification, RSA encryption and decryption, and HMAC authentication. We have implemented the Credential Manager component as a C++ Symbian server. The Credential Manager API allows installation and execution of custom credentials implemented as credential programs and provides convenience functions for common RSA operations. The Credential Manager implementation also orchestrates piecewise credential execution by maintaining intermediary sealed execution states and parameters.

The On-board Credentials TEE components and the Credential Manager are parts of Nokia Symbian devices starting from the Symbian Anna release. The deployed implementation covers credential provisioning and execution, OS-level application access control enforcements, and the key and application attestation mechanisms. For local user authentication, credential migration, temporary credential disabling and backup, we have research prototype implementations. For more details on the implementations of these components, see **P2**, **P4**, **P5**, and **P6**.

Credential programs for the On-board Credentials platform can be written in an assembler-style language that has a direct one-to-one mapping to the ObC bytecode set. Additionally, credential programs can be developed using a high-level programming language called EVO [55] which resembles Basic. Figure 4.2 illustrates a simple credential program written in EVO. As the application inputs, this credential program gets one plaintext data element and a previously provisioned credential secret in sealed format. The credential program encrypts the application data using the credential secret as the key and returns the resulting ciphertext as an output to the calling application. In our current implementation, the maximum size for a credential program is about one kilobyte in a compiled format. The exact upper

```
declare array data 10
declare array key 11
declare array cipher 12

#include "io_codes.evoh"
#include "program_io.evoh"
#include "aesenc.evoh"

function main()
    read_array(IO_PLAIN_RW, 0, data)
    read_array(IO_SEALED_RW, 1, key)
    aesenc(cipher, data, key)
    write_array(IO_PLAIN_RW, 0, cipher)
    return 0
end
```

Figure 4.2: Example encryption credential program

Credential type	Time
Encryption credential	30 ms
Signature credential	38 ms

Table 4.2: On-board Credentials platform performance measurements

limit depends on the amount of available on-chip secure memory on the target device. Larger credential instances can be created by combining multiple credential programs, i.e., endorsing several credential programs into the same credential family and executing them in a scheduled manner.

Two software development kits (SDKs) for On-board Credentials development are available. The third-party application development on Nokia Symbian devices is based on the Qt application framework. We have developed an extension to the Nokia Qt SDK [124] that allows development of Qt based applications that use the ObC system on Symbian phones. This SDK contains the needed headers and libraries for the Credential Manager API access, example code, and API documentation. Another SDK for credential program development is available. This SDK includes compilers from the EVO and assembly languages for the ObC bytecode and an emulation environment for testing and debugging credential programs on a PC before deployment to a TrustZone enabled mobile device. Both of these SDKs are available on request from the author.

Table 4.2 shows performance measurements from two example credentials. The encryption credential is similar to Figure 4.2. It performs an AES-EAX encryption for 16 bytes of data provided by the calling application with an encryption key that has been previously provisioned and is given to the credential program in sealed format. The overall execution time is 30 ms. This time includes (1) a process switch from the calling application to the Credential Manager server, (2) the Credential Manager pre-processing and a mode switch from TrustZone normal mode to secure mode, (3) the execution of the credential program on the Interpreter in secure mode, (4) a switch back to normal mode and Credential Manager post-processing, and (5) a process switch back to the calling application. The encryption credential was mea-

TEE type	Time
TrustZone (with ObC)	38 ms
Smart card	700–1550 ms
TPM	970 ms

Table 4.3: Performance comparison of 2048-bit RSA signatures

sured on a Nokia E7 device that runs on a 680-MHz ARM 11 processor. The reported execution time is averaged over 20 runs.

The signature credential calculates an RSA-SHA1 signature over 16 bytes of data using a 2048-bit RSA key. This signature credential was implemented using the RSA routines from the Credential Manager interface, i.e., without writing a new credential program. The overall execution time for an RSA signature is 38 ms with similar process and execution mode switch overheads included. The signature credential was tested on a device with comparable hardware features and the reported time is averaged over 20 runs. Compared to equivalent TPM signatures (approximately 970 ms [116]) and smart card signatures (700–1550 ms [36]), a solution that uses the TrustZone architecture has a significant performance advantage (see Table 4.3).

Generation of a 2048-bit RSA key takes five to ten seconds on a typical mobile device using the On-board Credentials platform. In terms of key generation performance, the main limiting factor is the limited amount of available on-chip secure memory. Commonly used prime generation optimization techniques cannot be implemented within such memory-constrained environments. Compared to equivalent TPM key generation (over twenty seconds [143]), ObC key generation is relatively fast.

Chapter 5

Discussion

5.1 Security summary

In this section, we summarize the security properties of the On-board Credentials platform. As already mentioned, we have alternative definitions for the TCB (TEE and OS) for the purpose of security analysis. The security mechanisms that rely on the TCB TEE are resistant to attackers with application installation, network control, OS compromise, simple off-line hardware attack and simple runtime hardware attack capabilities. The security mechanisms that rely on the TCB OS are vulnerable to OS compromise. Table 5.1 summarizes the attack resistance of the On-board Credentials system features.

All other provisioning requirements except user authentication hold against OS compromise. Since we assume no trusted user interface support from the TEE, user authentication during credential provisioning must be handled by OS-level code. A few possible approaches exist to address this limitation: (1) Mobile device manufacturers could configure user input and output components to the TrustZone secure mode. Since the implementation of the input and output drivers typically requires more memory than what is available on the on-chip memory components, this design choice would imply that also parts of the off-chip main memory should be configured to the secure mode, which would make the system vulnerable to physical attacks against the device main memory [84]. (2) The provisioning user authentication could be performed out-of-band (i.e., with another device), assuming that the enrollment process can be linked to the identity of the TEE reliably. In mobile devices such a suitable identifier is the IMEI (International Mobile Equipment Identifier). However, having to use a separate device and manually typing in the IMEI during provisioning can be inconvenient, or even impossible, in many use cases. (3) Provisioning operations could be limited to device boot when the device OS can be trusted due to the hardware-assisted secure boot. However, mandating a device boot before each credential installation can be inconvenient. In general, finding ways to perform more secure user authentication with current devices remains an open problem.

The TEE resource isolation, the OS isolation, the program isolation, and the computation limitation requirements rely on the TCB TEE. Since we assume no secure counters, the replay protection requirement relies on the integrity of the OS. Again, a few possible approaches exists to address this

Functionality	Requirement	TCB type
Provisioning	TEE authentication	TEE
	Provisioning user authentication	OS
	Program authorization	TEE
	Authorized sharing	TEE
	Program confidentiality	TEE
Execution	TEE resource isolation	TEE
	Program isolation	TEE
	OS isolation	TEE
	Computation limitation	TEE
	Replay protection	OS
	Runtime user authentication	OS
	Application authentication	OS
Life-cycle management	Authorized updates	TEE
	Controlled migration	TEE
	Temporary disabling	TEE
	Secure backups	OS
Attestation	Key attestation	TEE/OS
	Application attestation	OS

Table 5.1: Security summary of On-board Credentials features

limitation: (1) In many cases, the need for local credential state maintenance can be avoided with alternative credential design. In the case of public transport ticketing, for example, identity-based credentials could be used instead of stored-value credentials. (2) The server-based state protection model used in our credential migration solution could be extended into a more generic replay-protection mechanism. Performing an on-line freshness check before each credential execution may not be feasible. Some applications, for example, have very strict credential execution time requirements. In practice, credential freshness can be verified periodically, during device boot, or based on a credential-specific policy set by the credential issuer. For example, for high-security credentials such as payment system the freshness of the credential instance could be verified on-line before every credential invocation. Similar to provisioning user authentication also runtime user authentication relies on the OS. Application authentication is, by definition, an operating system feature, and thus it has to rely on the integrity of the platform security architecture.

Authorized updates, controlled migration and temporary credential disabling do not require OS integrity. In terms of attack resistance, replay protection for backups is equivalent to credential execution replay protection. Our key attestation is based on two distinct keys. As a result, compromise of the device OS does not compromise the attestation of TEE-internal access control properties. The application attestation mechanism builds on top of the runtime application identification, and thus relies on the integrity of the OS.

The On-board Credentials platform does not explicitly address denial-of-service attacks. We assume that the attacker can have physical possession of the device and thus can cause denial of service regardless of any preventive measures.

	Closed provisioning	Open provisioning
TEE authentication	pre-established domain	device certificate
Provisioning	pre-established domain	same-origin
Isolation	pre-verification or TEE isolation	TEE isolation
Runtime authentication	n/a	n/a
Migration	centralized	delegated
Updates	pre-established domain	same-origin
Disabling	n/a	n/a
Backups	n/a	n/a
Attestation	n/a	n/a

Table 5.2: Comparison between closed and open provisioning models

The On-board Credentials architecture is dependent on OS-level security mechanisms in many respects. This fact might raise the question that why hardware-assisted security mechanisms are used at all if the system relies on the integrity of the operating system? In many cases, the combination of OS-level and hardware-assisted security mechanisms provides the best possible overall attack resistance. In our application attestation model, for example, the applications identification is vulnerable to runtime OS attacks. But because the attestation key is protected by the TEE, our solution is resistant to simple off-line hardware attacks that would be possible if only software security mechanisms were used.

5.2 Open provisioning

Traditionally, hardware-based credential provisioning models have been closed and centralized. A closed provisioning model is typically justified with two arguments: First, closed provisioning allows business models in which service providers can be charged for credential installation. Second, a closed provisioning model enables better security and usability for credential installation and management. In this dissertation, we question the latter argument.

Table 5.2 summarizes the differences in the open and closed provisioning models. In closed provisioning, TEE authentication is usually based on a pre-established security domain (e.g., a shared symmetric key that is established during TEE manufacturing). In open provisioning, the identities of credentials issuers are not known at the time of TEE manufacturing but similar TEE authentication can be achieved with device certificates issued during manufacturing. In a closed model, the other provisioning requirements (program authorization and authorized sharing) are based on pre-established security domains while in open provisioning model similar features can be achieved by using dynamically established security domains and the same-origin policy.

In closed provisioning, the correctness of each credential program can be pre-verified by the credential issuer before provisioning. This is mandatory if the isolation properties on the TEE can be circumvented with malformatted bytecode, for example. In our architecture, such pre-verification is not needed. The ObC interpreter isolates credential programs regardless of the bytecode correctness. For an open provisioning model, an alternative is to ver-

ify the correctness of the bytecode within the TEE (e.g., many recent smart cards support on-card bytecode verification). Runtime application and user authentication implementations, on the other hand, are OS security architecture specific issues and independent from the provisioning model.

Regarding credential life-cycle management, closed provisioning has a noteworthy advantage. Credential migration is easy to implement in a closed model; all credentials can be re-provisioned, in a controlled manner, from the centralized credential issuer or TSM (see Section 2.3). In an open model, to achieve similar usability, a delegation mechanism with an external server must be used. The primary limitation of this approach is that the server has to be fully trusted, i.e., the credentials have to be encrypted for the server in such a way that the server can re-encrypt them for the target device. Two possible alternatives for addressing this drawback exist: (1) Proxy re-encryption [22] is a technique that allows an intermediary to re-encrypt data without having access to the plaintext data. The re-encryption is done with a key that is derived from the public part of the target device public key and the private part of the source device key. In credential transfer, the identity of the new (target) device may not be known before the user has to give away his old (source) device. Thus, proxy re-encryption cannot be easily applied to this kind of credential transfer. Additionally, proxy re-encryption requires the use of non-standard cryptographic primitives that are typically not available in the TEEs of current devices. (2) Credential backups could be protected with password based encryption. Techniques for retrieving password-encrypted data from a server without revealing the password to the server exist [25]. Also these techniques require the use of non-standard cryptography.

In both models, authorized credential updates are easy to facilitate (pre-established security domains in the closed model and the same-origin policy in the open model). Temporary credential disabling is independent from the provisioning model. If the TEE does not support counters, then an external assisting server is needed. Backups, attestation and replay protection are independent from the provisioning model.

We conclude that the primary benefit of closed provisioning is in credential migration. Assuming that an external and trusted server can be used to assist in credential life-cycle management operations, comparable levels of security and usability can be achieved with an open provisioning model as well. The mobile platform providers are in a suitable position to provide such trusted services. The existing mobile platform provider on-line services such as application stores could be extended to support services like credential migration. At the same time, we acknowledge that for certain credential issuers (e.g., financial services) dependence to a third-party authority may not be acceptable.

In many use cases, the best balance between security, usability and flexibility can be achieved with a *hybrid model*. Credentials originating from well-known major service providers can be provisioned and managed centrally with pre-established security domains while open provisioning and management with the same-origin policy can be allowed for credentials originating from unknown service providers. In such a setup, the TEE should distinguish between these two credential types based on the provisioning security domain and enforce credential management policies (e.g., migration) accordingly. For more detailed comparison between the open and closed models, see **P7**.

5.3 Related work

Several topics related to this dissertation are discussed in a comprehensive survey [127].

TrustZone. MobiCore [71] is a recent commercial credential platform that allows development of credentials that are executed within TrustZone TEE. The isolation is based on a small operating system, or microkernel, that isolates credentials from the TEE resources. The details of the MobiCore system are not public, but presumably it requires more runtime memory than the ObC Interpreter, which implies that parts of the off-chip main memory must be configured to the TrustZone secure mode. The MobiCore provisioning model is closed; all installed credentials must go through a central controlling entity.

Smart cards. Google Wallet [78] is another commercial credential system for smartphones. The primary purpose of Google Wallet is to enable integration of payment credentials to NFC-enabled smartphones. The TEE in the Google Wallet system is a smart card that is embedded to the mobile device. The provisioning model of the credentials is a typical closed smart card provisioning, i.e., deployment of new Google Wallet credentials requires a permission from the platform provider.

Trusted Execution Module (TEM) [38] is a credential system intended for smart cards with an open provisioning model. In the TEM system, credential program authorization and authorized sharing are based on memory addresses. A credential program can access a secret from or write a new secret to a memory address that unauthorized credential programs do not know and thus cannot access. The fact that a credential program knows a secret memory address guarantees that it originates from the original issuer (or another issuer authorized by the original issuer). For a more detailed comparison, see **P2**. User-centric smart card provisioning models and architectures have been discussed by Akram [2].

In model-carrying code system [145], the code producer creates a model that captures the security-relevant operations made by the code and the code consumer verifies that this model complies with a device-specific policy using either static analysis or runtime enforcement. This concept has been proposed to control interactions between smart card applications [50].

Late launch. On PC platforms, the late launch features of the latest processors can be used to provide a secure execution environment for security-critical code. Flicker [115, 116] provides an isolated execution environment with a minimal Trusted Computing Base. In the Flicker system, security-critical code is measured, the measurement is extended to a TPM register, the code is then executed in isolation from rest of the system, and later an attestation of the code can be provided to an external verifier. Depending on the processor architecture and the size of the executed code, either the processor cache or the device main memory is used for the execution. The executed code may permanently store data using TPM sealing, and the freshness of the seals can be protected with TPM counters. Safety and liveness properties of state continuity in such execution environments is studied in more detail in [126].

Unicorn [112] is a system that utilizes a combination of a late launch execution on a PC and a trusted mobile device for attestation verification. In the Unicorn system, the late launched security sensitive code is measured and executed, and the resulting attestation is sent to the mobile device for verification. The Unicorn security model is based on the assumption that the attacker is not able to compromise both the PC and the mobile device at the same time. Bumpy [117] is a system that allows secure user input from external devices to late launched execution environments. User verification of local TPM identity is discussed in [125].

Virtualization. Virtualization technology is another way of providing an isolated execution environment for security critical processing [70]. In the virtualization approaches, security sensitive and untrusted software components are separated into different operating system instances, or compartments. The OS compartments are isolated from one another by a hypervisor. The popular Xen hypervisor has been ported to the ARM architecture [66], and more recently mobile virtualization has been addressed, for example, in [146, 43, 105]. Some of these approaches provide mechanisms for trusted user input [146, 43]. On PC platforms, virtualization technology has been used in combination with TPM-based protection for better attack resistance [114]. The isolation properties of some hypervisors, or microkernels, have been formally verified [98]. Similar to virtualization, *hyperpartioning* [158] can provide separation of trusted and untrusted software components.

Mobile Trusted Module (MTM). The MTM specifications define an interface, rather than a hardware implementation. The device manufacturers are allowed to implemented the MTM interface in different ways. Research prototype implementations based on ARM TrustZone [164, 56], smart cards [48], and late launch [29] have been reported. However, so far Mobile Trusted Module implementations have not been widely deployed in practice.

Hardware architectures. Researchers have also proposed new types of hardware architectures for secure storage and execution of security-critical code. An architecture based on two security chips: one with high-speed processing and another with a persistent secure state is outlined in [37]. Minimal hardware updates (new hardware registers and instructions) that are needed to support secure storage and execution in an authority-based trust model have been discussed in [52, 106]. The current TrustZone based mobile devices implement many of these concepts in practice. Secure processor architecture based on physically unclonable functions is discussed in [153]. Cryptographic coprocessors such as the IBM 4758 cryptocard are used in some security-critical on-line servers. In this dissertation, we focus on TEEs supported by the existing commodity end-user devices.

Life-cycle management. Prior to our work, credential transfer between trusted execution environments has been studied primarily in the context of TPMs [69, 102, 21, 34, 137]. None of the previous publications addresses user-friendly re-provisioning of non-migratable credentials or secure user binding in devices with no trusted user interface. To the best of the author's knowledge, our work is the first to address the problem of temporary credential disabling on devices without secure non-volatile memory. The idea of utilizing an ex-

ternal server as the source of secure state is known [157]. For more detailed comparison to related work see **P3**, **P5** and **P7**.

Attestation. The previous key attestation mechanisms report only TEE-internal credential access properties [152] or TEE-external application access control settings [80]. Our key attestation mechanism combines these two concepts in such a way that the possible OS compromise does not affect TEE-internal attestations. Property-based attestation has been studied extensively in the past [136, 131, 31, 32, 100, 94, 121]. Our attestation mechanism is the first that bootstraps from existing application signing infrastructures and, thus, makes realization of property-based attestation in the context of mobile devices feasible to deploy. Independent of our work, an attestation mechanisms based on the Android platform security architecture has been presented [20]. For more detailed comparison to related attestation work, see **P4** and **P6**.

Platform security. Many open issues in mobile OS security still remain. Very fine-grained permissions cause developer and end-user confusion while a coarse-grained permission model violates the principle of least privilege [13, 64]. Application signing and vetting by a centralized authority limits the freedom of application developers while developer signing assumes that end-users are capable and willing to determine if a certain permission should be granted to an application—which most of the time is not the case. Colluding applications may circumvent permission-based access control models [65, 113]. To address these problems, researchers have recently proposed methods for identifying malicious applications based on the permissions they request [61, 62], applying data and control flow techniques [42], hardening the mobile device operating system with mandatory access control features [28, 27, 33, 147], and more flexible permission models [120]. For more detailed discussion on recent developments on mobile platform security we refer the reader to recent surveys [58, 17].

5.4 Trusted execution environments

In this section, we compare common trusted execution environments (see Table 5.3). We start with credential development, performance and certification. In smart cards, credentials must be fairly small, the development is based on widely used JavaCard tools, and the performance is typically poor due to low-speed processor. TrustZone offers better performance thanks to the use of the device main processor, but credential sizes are very limited assuming only on-chip secure memory, and the development is based on custom tools. Late launch based solutions allow unlimited credential sizes if device main memory is used. Otherwise, late launch credentials are limited by the processor cache size. The use of slow TPM signatures imposes a considerable performance overhead in late launch systems. In virtualization based credential systems, there are no noteworthy development or performance restrictions.

Traditionally, only smart cards and TPMs have achieved commonly accepted security certification levels. Recently, also other types of security environments (e.g., MobiCore [71]) have been certified. Virtualization based security environments have been formally verified [98]. Lack of formal certi-

	Smart card	**TrustZone**	**Late launch**	**Virtualization**
Credential systems	TEM [38], Google Wallet [78]	ObC, MobiCore [71], MTM [164]	Flicker [115, 116], MTM [29], Unicorn [112], Bumpy [117]	TVD [43], TMD [146], TrustVisor [114]
TEE type	removable or embedded	embedded	embedded	embedded
Credential size	tens of kilobytes	few kilobytes	unlimited (main memory) or small (processor cache)	unlimited
Performance	low (low-speed processor)	high (main processor)	low (TPM signatures)	high (main processor)
Development tools	JavaCard	custom	custom	any
Security certification	yes	typically no	typically no (TPM is certified)	typically no
OS compromise	resistant	resistant	resistant	resistant
Off-line attack	resistant	resistant	resistant	vulnerable
Simple runtime hardware attack	resistant	resistant (on-chip memory) or vulnerable (off-chip memory)	resistant (processor cache) or vulnerable (main memory)	vulnerable
Sophisticated hardware attack	resistant	vulnerable	vulnerable	vulnerable
Replay protection	yes	configuration issue (typically no)	yes	no
Trusted user interface	no	configuration issue (typically no)	possible	possible
TCB	small (e.g., JavaCard VM)	small (e.g., ObC Interpreter)	minimal	large (hypervisor)

Table 5.3: Comparison of trusted execution environments

fication can be a significant problem for deployment in financial services, for example. While we acknowledge the benefits of thorough security evaluation and certification, it should be noted that the current security certification systems do not guarantee an error-free system. The Common Criteria certification, for example, suffers from wrong incentives [51] and certified systems have been broken [119].

In terms of security, smart cards offer the best protection against various attacks. Credentials implemented using TrustZone are resistant to most hardware attacks, excluding sophisticated runtime attacks. The attack resistance of late launch systems depends on the processor architecture. If the device main memory is used, such systems can be compromised by launching hardware attacks against the main memory. The same argument holds for TrustZone as well if parts of the off-chip memory are configured to the TrustZone secure mode. Out of these approaches, virtualization offers the lowest attack resistance and solutions based on pure virtualization can be compromised with simple off-line attacks. To mitigate this threat, some solutions combine TPM-based off-line protection with virtualization [114]. Smart cards and TPM systems support secure non-volatile memory and enable hardware-based replay protection. In TrustZone, on-device replay protection is a matter of hardware configuration and typically not available.

Smart cards do not provide any user input and output mechanism. In a typical mobile device configuration, user input and output are not parts of the TrustZone secure mode either. Virtualization allows implementation of user input and output mechanisms that are isolated from the untrusted parts of the system. In recently proposed approaches [146, 43], the trusted user interface is based on visual security indicators which aim to ensure that the user only inputs security sensitive data (e.g., passwords) into the trusted part of the system. The effectiveness of such indicators is questionable; we discuss trusted user input more in Section 5.7.

Late launch technology enables credential systems with a minimal trusted computing base. In smart card and TrustZone system, small isolation environments (e.g., the ObC Interpreter or the JavaCard virtual machine) must be trusted. In virtualization solutions, an entire hypervisor must be trusted.

5.5 Applications

In this section, we discuss promising application areas for the On-board Credentials technology.

- *Public transport ticketing* with NFC-enabled smartphones is a promising use case for our system. Ticketing protocols and architectures using the ObC platform are discussed in detail in [57, 154]. In transport ticketing, the main challenges arise from user experience requirements. The exact requirements vary between systems and use cases but, as an illustrative example, an NFC-based user authentication protocol at a public transport gate should take no longer than 300 milliseconds [3]. Meeting such time limits excludes the use of conventional cryptographic primitives and protocols, and novel protocols are needed in many cases. With optimized protocols, ObC-based ticketing has been shown feasible [57, 154]. The New York Metropolitan Transport Authority (MTA)

is currently starting a trial with ObC ticketing credentials and NFC smartphones [123].

- *Physical access control* is another interesting application area. Example use cases include opening home, office and car doors with NFC devices. Compared to traditional keys and contactless access token, smartphones with rich user interfaces and Internet connectivity enable new and interesting possibilities like delegating physical access rights from one device to another [16].

- *Payments.* The lack of security certification in TrustZone TEE limits the applicability of the ObC platform for high-value payments. Especially credit card companies seem to consider certification a mandatory requirement. Novel types of NFC-enabled proximity payments [96] and payment systems for developing regions [149] are potential application areas where the ObC system could be used for low-value financial applications. Additionally, the one-time password based user authentication schemes used by many banks could be easily implemented with ObC. Implementation experiences of an ObC-based one-time password system are presented in [160].

- *Token replacement.* ObC enabled smartphones can also replace traditional security tokens. Our collaborators at RSA Inc. have implemented their proprietary SecurID authentication algorithm as an ObC program. This project was an example of a real-world credential program that requires confidentiality. A mobile device with the On-board Credentials platform could also act as a generic replacement for smart cards [11, 155].

- *Web credentials.* Publication **P8** presents an ObC-based system for the storage of web authentication passwords. If novel web authentication systems, such as BrowserID [103, 89], get widely adopted, the ObC system would provide a good implementation platform for the secure storage and execution of such web credentials.

- The *property-based attestation* mechanism that we have presented in this dissertation is another use case for the On-board Credentials system. The attestation mechanism is now part of the MirrorLink standard [53] and the first product implementations are on-going. We expect to see many mobile devices and cars supporting this attestation mechanism during the next few years. The attestation mechanism itself, however, is not specific to MirrorLink. Similar application attestation could be applied in other use cases as well.

Various other potential use cases exist in addition to the ones listed above. The possibility of implementing SIM cards (Subscriber Identity Module) without physical smart cards has attracted considerable attention recently [19]. The On-board Credentials platform would provide an ideal foundation for such implementations. Also content-protection services like digital rights management could benefit from the ObC platform. Finally, probably the *greatest feature* of the ObC platform is the fact that developers can create *arbitrary types of credentials*. Thus, we expect to see many credential types besides the ones listed here appearing in the near future.

5.6 Portability

We have designed and implemented the On-board Credentials platform for TrustZone mobile devices with the Symbian OS. In this section, we discuss how the ObC architecture can be adapted to other environments.

While in most respects the ObC architecture is OS-agnostic, there are a few noteworthy aspects that should be considered when porting the ObC system to another OS. In some smartphone platforms, applications are self-signed by developers (e.g., Android). In such platforms, the application identification needed in our property-based attestation scheme must be a combination of developer identification (e.g., a hash of the public part of the application signing key) and a developer-assigned application identifier (e.g., the package name).

In the Symbian OS, system components are typically implemented as system servers that have unique identifiers assigned by the central signing authority. In the Symbian platform, also third-party developers can implement and deploy system servers. In most other smartphone platforms, unique identifiers are assigned only to downloaded third-party applications and third-party development is limited to applications. This affects the ability to identify and, based on such identification, perform access control or attestation on the OS level. Especially, in the case of our property-based attestation mechanisms, this affects the ability to attest system components separately. In platforms that allow only manufacturer provided system components, this limitation is not a significant disadvantage; all system components can be trusted by default, because they originate from the manufacturer.

With these minor aspects in mind, we argue that the On-board Credentials architecture can be ported to any other prominent mobile operating system. Adaptations to other major mobile platforms are current work in progress.

The ObC system could be implemented on other types of TEEs as well. An early TPM-based ObC architecture adaptation is described in [150]. In this design, the ObC Interpreter is implemented as a kernel module. The integrity and secure storage of this kernel module are enforced with a TPM-style authenticated boot process. A natural extension to this work would be to utilize the security features of the latest PC processors and to execute the ObC Interpreter and the other ObC TEE components using a late launch trusted execution environment such as Flicker [116]. Instead of porting the ObC system as it is to smart cards, a more interesting approach would be to adapt the JavaCard platform to support the principles of open provisioning. Instead of pre-established security domains, the JavaCard platform should be modified to allow dynamically established security domains based on the same-origin policy.

The mobile device landscape is evolving and new forms of TEEs are emerging. GlobalPlatform is currently defining standardized TEE features and interfaces for mobile devices [76, 75, 72]. One potential implication of these new standards is that integrated hardware security architectures on mobile devices can in the future provide security certification levels (e.g., following procedures outlined in [73]). Also, the Trusted Computing Group is currently working on the next version of the Mobile Trusted Module (MTM) standard [82]. How exactly these emerging standards fit together, remains to be seen. Figure 5.1 illustrates one possible approach. The On-board Credentials plat-

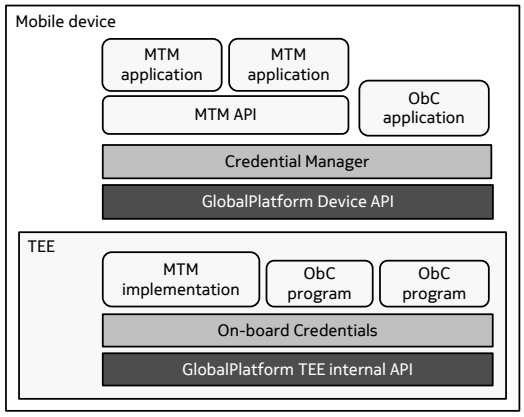

Figure 5.1: Possible architecture for emerging TEE standards

form could be implemented using the GlobalPlatform interface specifications while ObC could provide the foundation for flexible implementation of the MTM standard. We plan to implement the MTM standard as an ObC credential program to validate the feasibility of this approach.

5.7 Future work

We have formally modeled and verified certain parts of the On-board Credentials system, i.e., a subset of the credential transfer protocol. An obvious continuation for this work would be to verify the rest of the system as well. Security protocols could be verified with tools like AVISPA [159]. Modeling and verification of security protocols that rely on correct user interaction is a topic with little existing work and an interesting direction for further research. Security verification of the isolation properties of the ObC Interpreter could follow the principles used in the verification of smart card and JavaCard platforms [156, 93, 14, 6, 7, 142]. Other security properties of the On-board Credentials system design could be formally modeled and potentially verified using verification tools and languages such as TLA+ Proof System [30].

In many use cases, traditional security protocols may not be applicable or sufficient. The proximity-based wireless communication channels and the secure storage and processing capabilities in current devices are limited while the user experience requirements can be very strict. Public transport ticketing is an example of a use case in which novel protocols are needed [154, 57]. The perceived speed and convenience are critical to user satisfaction and acceptance [15]. One possible approach is to split the security-critical communication and processing into two parts: (1) communication that must happen in real-time between the entities in proximity of each other and (2) the non-time-critical communication and processing that can be performed before or after the proximity-based transaction on-line with an infrastructure backend. Such protocol design is another avenue for further research.

Users have established mental models on how services like physical access control, ticketing and payments work with traditional credentials. Implement-

ing these services with virtual credentials and improving the existing services with the possibilities provided by the current mobile devices (e.g., rich user interfaces and Internet connectivity), are likely to cause user confusion. Trials and user studies are needed to find the best possible usage models [15]. In cases where usability and security requirements seem to contradict each other, the optimal balance between security and usability needs to be studied carefully. The credential life-cycle management operations discussed in this dissertation require user interaction as well. Without thorough user studies, the effectiveness of these proposed models remains uncertain.

The lack of trustworthy user input mechanism is one of the fundamental problems in trusted computing. Research prototypes, mainly based on virtualization [146, 43], provide user input and output that is isolated from the untrusted parts of the device. These approaches are based on visual indicators such as colored toolbars for trusted and untrusted parts of the system. Several studies have shown that users tend to ignore visual security indicators [140, 40, 165, 54]. User training [151] and interactive security indicators [109] are potential ways to improve the effectiveness of trusted user interfaces. In general, finding secure user input and output mechanisms that do work in practice and can be implemented using existing devices remains an open research problem.

Finally, the On-board Credentials development model requires that the developers isolate their security critical processing from the rest of the application and implement it as an ObC credential program while the rest of the needed processing (e.g., network communication and user interaction) is handled by the OS-level application. The task of the developers could be made easier by developing automated tools for such processing separation [95, 26].

Chapter 6

Conclusions

In this dissertation, we have described several aspects of a novel credential platform called On-board Credentials. The ObC system can be used to implement credentials that are secure, usable and inexpensive at the same time. The distinguishing feature of the On-board Credentials platform is the open provisioning model that allows any service provider to develop and issue new credential types and instances without having to request permission from a controlling authority.

Our primary implementation target has been Symbian smartphones with the TrustZone security architecture. This work has shown that implementation of a fully functional credential platform is possible even within the severe limitations of the current TrustZone mobile devices. Our implementation is part of the latest Nokia Symbian devices. While some of our design decisions are driven by the specifics of our target environment, most of the concepts presented in this dissertation are applicable to other kinds of platforms as well. The On-board Credentials system is currently being ported to other mobile platforms.

In this dissertation, we have addressed the entire life-cycle of hardware-protected credentials. We have presented solutions for secure credential provisioning, storage and execution, credential issuer updates and end-user management. The key learnings of this dissertation can be summarized as follows:

- Our open provisioning model with the *same-origin policy* provides a viable alternative for traditional closed provisioning. We show that, as opposed to a common view, secure credential provisioning and subsequent credential management is possible even in open credential platforms.

- The openness of the credential platform complicates certain credential life-cycle management operations. Credential migration, for example, is easier to implement in a closed and centralized credential system. We show that using a *delegated credential re-provisioning* mechanism, user-friendly credential migration can be realized also in open platforms (assuming a trusted third-party).

- Also the limitations of the existing trusted execution environments cause challenges. First, the lack of trusted user input affects credential migration. We show that by applying the principle of *trust on first use*,

secure credential migration can be achieved even with devices without trusted user interfaces. Second, the lack of secure counters complicates temporary credential disabling and replay protection in general. We have presented a credential disabling solution that mitigates this limitation by using an *on-line server as the trusted state*. Third, the limited amount of secure on-chip memory on current mobile devices makes isolated execution of arbitrary credentials difficult. The ObC architecture provides a custom bytecode language and a minimal interpreter that allows secure credential execution even in very resource-constrained environments supported by the existing mobile devices.

– The current platform security architectures and application signing mechanisms provide a powerful foundation for building various security solutions. In this dissertation, we have presented a *novel property-based attestation* mechanism that bootstraps application authentication from existing mobile application signing infrastructures. Our model makes realization of property-based attestation practical with current mobile devices.

– Public transport ticketing, physical access control and proximity payments are promising application areas for the ObC system. However, potentially the greatest power of the ObC system is the fact that it allows *anybody* to develop *arbitrary* credential types. The feasibility of various ObC credentials have already been demonstrated, and quite likely, the most fascinating applications are ones that we, the designers of the ObC platform, cannot predict.

We argue that this work is a significant step towards a vision of using a smartphone as a *personal trusted device* that integrates a wide range of credentials in a user-friendly, secure and inexpensive manner. The first noteworthy credential deployments are now starting, and we expect to see considerable real-world use of the On-board Credentials platform during the next few years.

Bibliography

[1] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Communication of the ACM*, 42(12):40–46, December 1999.

[2] Raja Naeem Akram, Konstantinos Markantonakis, and Keith Mayes. A paradigm shift in smart card ownership model. In *Proceedings of the International Conference on Computational Science and Its Applications (ICCSA)*, pages 191–200, March 2010.

[3] Smart Card Alliance. Transit and contactless financial payments: New opportunities for collaboration and convergence: A Smart Card Alliance transportation council white paper, October 2006. Available at: http://www.smartcardalliance.org/.

[4] James Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, Electronic Systems Division, October 1972.

[5] Ross Anderson, Mike Bond, Jolyon Clulow, and Sergei Skorobogatov. Cryptographic processors: A survey. *Proceedings of the IEEE*, 94(2):357–369, February 2006.

[6] June Andronick, Boutheina Chetali, and Olivier Ly. Using Coq to verify JavaCard applet isolation properties. In *Proceedings of the Theorem Proving in Higher Order Logic (TPHOL)*, pages 335–351, September 2003.

[7] June Andronick, Boutheina Chetali, and Christine Paulin-Mohring. Formal verification of security properties of smart card embedded source code. In *Proceedings of the International Symposium on Formal Methods (FM)*, pages 302–317, July 2005.

[8] William A. Arbaugh, David J. Farber, and Jonathan M. Smith. A secure and reliable bootstrap architecture. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 65–71, May 1997.

[9] ARM. ARM security technology: Building a secure system using TrustZone technology, April 2009. Available at: http://www.arm.com/.

[10] Jerome Azema and Gilles Fayad. M-shield mobile security technology: making wireless secure, February 2008. Available at: http://www.ti.com/.

[11] Dirk Balfanz and Edward W. Felten. Hand-held computers can be better smart cards. In *Proceedings of the USENIX Security Symposium*, August 1999.

[12] Shane Balfe, Eimear Gallery, Chris J. Mitchell, and Kenneth G. Paterson. Challenges for trusted computing. *IEEE Security Privacy*, 6(6):60–66, November–December 2008.

[13] David Barrera, H. Günes Kayacik, Paul C. van Oorschot, and Anil Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In *Proceedings of the ACM conference on Computer and Communications Security (CCS)*, pages 73–84, October 2010.

[14] Gilles Barthe, Guillaume Dufay, Line Jakubiec, Bernard P. Serpette, and Simao Melo de Sousa. A formal executable semantics of the JavaCard platform. In *Proceedings of the European Symposium on Programming Languages and Systems (ESOP)*, pages 302–319, April 2001.

[15] Lujo Bauer, Lorrie Faith Cranor, Michael K. Reiter, and Kami Vaniea. Lessons learned from the deployment of a smartphone-based access-control system. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*, pages 64–75, July 2007.

[16] Lujo Bauer, Scott Garriss, Jonathan M. Mccune, Michael K. Reiter, Jason Rouse, and Peter Rutenbar. Device-enabled authorization in the Grey system. In *Proceedings of the Information Security Conference (ISC)*, pages 431–445, September 2005.

[17] Michale 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 IEEE Symposium on Security and Privacy (SP)*, pages 96–111, May 2011.

[18] Mihir Bellare, Phillip Rogaway, and David Wagner. The eax mode of operation: A two-pass authenticated-encryption scheme optimized for simplicity and efficiency. In *Proceedings of the Fast Software Encryption workshop (FSE)*, pages 389–407, February 2004.

[19] Diana ben Aaron. Bloomberg: GSMA explores software-based replacement for mobile SIM cards, November 2010. http://www.bloomberg.com/news/2010-11-18/gsma-explores-software-based-replacement-for-mobile-sim-cards.html. Referenced: December 2011.

[20] Ingo Bente, Gabi Dreo, Bastian Hellmann, Stephan Heuser, Joerg Vieweg, Josef von Helden, and Johannes Westhuis. Towards permission-based attestation for the Android platform. In *Proceedings of the International Conference on Trust and trustworthy computing (TRUST)*, pages 108–115, June 2011.

[21] Stefan Berger, Ramon Caceres, Kenneth A. Goldman, Ronald Perez, Rainer Sailer, and Leendert van Doorn. vTPM – virtualizing the Trusted

Platform Module. In *Proceedings of the USENIX Security Symposium*, pages 305–320, July 2006.

[22] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 127–144, May 1998.

[23] Hristo Bojinov, Dan Boneh, Rich Cannings, and Iliyan Malchev. Address space randomization for mobile devices. In *Proceedings of the ACM conference on Wireless network Security (WiSec)*, pages 127–138, June 2011.

[24] Raja Bose, Jörg Brakensiek, Keun-Young Park, and Jonathan Lester. Morphing smartphones into automotive application platforms. *IEEE Computer*, 44(5):53–61, May 2011.

[25] Xavier Boyen. Hidden credential retrieval from a reusable password. In *Proceedings of the International Symposium on Information Computer and Communications Security (ASIACCS)*, pages 228–238, March 2009.

[26] David Brumley and Dawn Song. Privtrans: Automatically partitioning programs for privilege separation. In *Proceedings of the USENIX Security Symposium*, pages 57–72, August 2004.

[27] Sven Bugiel, Lucas Davi, Alexandra Dmitrienko, Thomas Fischer, Ahmad-Reza Sadeghi, and Bhargava Shastry. Towards taming privilege-escalation attacks on Android. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, February 2012.

[28] Sven Bugiel, Lucas Davi, Alexandra Dmitrienko, Stephan Heuser, Ahmad-Reza Sadeghi, and Bhargava Shastry. Practical and lightweight domain isolation on Android. In *Proceedings of the ACM workshop on Security and Privacy in Smartphones and Mobile devices (SPSM)*, pages 51–62, October 2011.

[29] Sven Bugiel and Jan-Erik Ekberg. Implementing an application-specific credential platform using late-launched Mobile Trusted Module. In *Proceedings of the ACM workshop on Scalable Trusted Computing (STC)*, pages 21–30, October 2010.

[30] Kaustuv Chaudhuri, Damien Doligez, Leslie Lamport, and Stephan Merz. Verifying safety properties with the TLA+ proof system. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR)*, pages 142–148, July 2010.

[31] Liqun Chen, Rainer Landfermann, Hans Löhr, Markus Rohe, Ahmad-Reza Sadeghi, and Christian Stüble. A protocol for property-based attestation. In *Proceedings of the ACM workshop on Scalable Trusted Computing (STC)*, pages 7–16, November 2006.

[32] Liqun Chen, Hans Löhr, Mark Manulis, and Ahmad-Reza Sadeghi. Property-based attestation without a trusted third party. In *Proceedings of the Information Security Conference (ISC)*, pages 31–46, September 2008.

[33] Google Code. TOMOYO Linux on Android. http://code.google.com/p/tomoyo-android/. Referenced: December, 2011.

[34] Andrew Cooper and Andrew Martin. Towards an open, trusted digital rights management platform. In *Proceedings of the ACM workshop on Digital Rights Management (DRM)*, pages 79–88, October 2006.

[35] Intel Corporation. Intel trusted execution technology (Intel TXT): Software development guide, March 2011. Available at: http://www.intel.com/.

[36] Victor Costan. JCOP smartcard performance, November 2009. http://blog.costan.us/2009/11/jcop-smartcard-performance.html. Referenced: December 2011.

[37] Victor Costan and Srinivas Devadas. Security challenges and opportunities in adaptive and reconfigurable hardware. In *Proceedings of the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 1–5, June 2011.

[38] Victor Costan, Luis Sarmenta, Marten van Dijk, and Srinivas Devadas. The trusted execution module: Commodity general-purpose trusted computing. In *Proceedings of the Smart Card Research and Advanced Application Conference (CARDIS)*, pages 133–148, September 2008.

[39] European Payments Council and GSM Association. Trusted service manager service management requirements and specifications, January 2010. Available at: http://www.europeanpaymentscouncil.eu/.

[40] Lorrie Faith Cranor. What do they "indicate?": evaluating security and privacy indicators. *Interactions*, 13:45–47, May 2006.

[41] Common Criteria. Common criteria for information technology security evaluation, part 1: Introduction and general model, July 2009. Available at: http://www.commoncriteriaportal.org/.

[42] Lucas Davi, Alexandra Dmitrienko, Manuel Egele, Thomas Fischer, Thorsten Holz, Ralf Hund, Stefan Nrnberger, and Ahmad-Reza Sadeghi. MoCFI: A framework to mitigate control-flow attacks on smartphones. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, February 2012.

[43] Lucas Davi, Alexandra Dmitrienko, Christoph Kowalski, and Marcel Winandy. Trusted virtual domains on OKL4: secure information sharing on smartphones. In *Proceedings of the ACM workshop on Scalable Trusted Computing (STC)*, pages 49–58, October 2011.

[44] Jack B. Dennis and Earl C. Van Horn. Programming semantics for multiprogrammed computations. *Communications of the ACM*, 9:143–155, March 1966.

[45] CVE Details. The ultimate security vulnerability resource. http://www.cvedetails.com/. Referenced: December, 2011.

[46] Damien Deville and Gilles Grimaud. Building an "impossible" verifier on a JavaCard. In *Proceedings of the Conference on Industrial Experiences with Systems Software (WIESS)*, pages 16–24, December 2002.

[47] Rachna Dhamija, J. D. Tygar, and Marti Hearst. Why phishing works. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pages 581–590, April 2006.

[48] Kurt Dietrich and Johannes Winter. Towards customizable, application specific Mobile Trusted Modules. In *Proceedings of the ACM workshop on Scalable Trusted Computing (STC)*, pages 31–40, October 2010.

[49] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, March 1983.

[50] Nicola Dragoni, Olga Gadyatskaya, and Fabio Massacci. Can we support applications evolution in multi-application smart cards by security-by-contract? In *Proceedings of the Workshop in Information Security Theory and Practice (WISTP)*, pages 221–228, April 2010.

[51] Saar Drimer, Steven J. Murdoch, and Ross Anderson. Thinking inside the box: System-level failures of tamper proofing. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 281–295, May 2008.

[52] Jeffrey Dwoskin and Ruby Lee. Hardware-rooted trust for secure key management and transient trust. In *Proceedings of the ACM Conference on Computer and Communication Security (CCS)*, pages 389–400, October 2007.

[53] Car Connectivity Consortium (editor Jörg Brakensiek). Terminal Mode technical architecture: version 1.0, October 2011. Available at: http://carconnectivity.org/.

[54] Serge Egelman, Lorrie Faith Cranor, and Jason Hong. You've been warned: an empirical study of the effectiveness of web browser phishing warnings. In *Proceedings of the Conference on Human factors in computing systems (CHI)*, pages 1065–1074, April 2008.

[55] Jan-Erik Ekberg. The EVO language external and internal overview, October 2011. Personal communication, available on request.

[56] Jan-Erik Ekberg and Sven Bugiel. Trust in a small package: minimized MRTM software implementation for mobile secure environments. In *Proceedings of the ACM workshop on Scalable Trusted Computing (STC)*, pages 9–18, November 2009.

[57] Jan-Erik Ekberg and Sandeep Tamrakar. Mass transit ticketing with NFC mobile phones. In *Proceedings of the International Conference on Trusted Systems (INTRUST)*, December 2011.

[58] William Enck. Defending users against smartphone apps: Techniques and future directions. In *Proceedings of the International Conference on Information Systems Security (ICISS)*, pages 49–70, December 2011.

[59] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 393–408, October 2010.

[60] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. A study of Android application security. In *Proceedings of the USENIX Security Symposium*, pages 315–330, August 2011.

[61] William Enck, Machigar Ongtang, and Patrick McDaniel. On lightweight mobile phone application certification. In *Proceedings of the ACM conference on Computer and Communications Security (CCS)*, pages 235–245, November 2009.

[62] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the ACM Conference on Computer and Communication Security (CCS)*, pages 627–638, October 2011.

[63] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. A survey of mobile malware in the wild. In *Proceedings of the ACM workshop on Security and Privacy in Smartphones and Mobile devices (SPSM)*, pages 3–14, October 2011.

[64] Adrienne Porter Felt, Kate Greenwood, and David Wagner. The effectiveness of application permissions. In *Proceedings of the USENIX Conference on Web Application Development (WebApps)*, pages 75–86, June 2011.

[65] Adrienne Porter Felt, Helen Wang, Alex Moshchuk, Steve Hanna, and Erika Chin. Permission re-delegation: Attacks and defenses. In *Proceedings of the USENIX Security Symposium*, pages 331–346, August 2011.

[66] Daniel R. Ferstay. Fast secure virtualization for the ARM platform. Master's thesis, University of British Columbia, March 2006.

[67] Thomas Fischer, Ahmad-Reza Sadeghi, and Marcel Winandy. A pattern for secure graphical user interface systems. In *Proceedings of the International workshop on Database and Expert Systems Application (DEXA)*, pages 186–190, August 2009.

[68] NFC Forum. Essentials for successful NFC mobile ecosystems, October 2008. Available at: http://www.nfc-forum.org/.

[69] Sebastian Gajek, Hans Löhr, Ahmad-Reza Sadeghi, and Marcel Winandy. TruWallet: trustworthy and migratable wallet-based web authentication. In *Proceedings of the ACM workshop on Scalable Trusted Computing (STC)*, pages 19–28, November 2009.

[70] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: a virtual machine-based platform for trusted computing.

In *Proceedings of the ACM symposium on Operating Systems Principles (SOSP)*, pages 193–206, October 2003.

[71] Giesecke and Devrient. Mobicore: Giesecke and Devrient's secure OS for ARM TrustZone technology, 2011. Available at: http://www.gi-de.com/.

[72] GlobalPlatform. Device technology TEE client API specification: version 1.0, July 2010. Available at: http://www.globalplatform.org/.

[73] GlobalPlatform. Card composition model: version 1.1.0.6, November 2011. Available at: http://www.globalplatform.org/.

[74] GlobalPlatform. Card specification: version 2.2.1, January 2011. Available at: http://www.globalplatform.org/.

[75] GlobalPlatform. Device technology TEE internal API specification: version 0.27 public draft, September 2011. Available at: http://www.globalplatform.org/.

[76] GlobalPlatform. Device technology TEE system architecture: version 0.4 public draft, October 2011. Available at: http://www.globalplatform.org/.

[77] GlobalPlatform. System messaging specification for management of mobile NFC services: version 1.0, January 2011. Available at: http://www.globalplatform.org/.

[78] Google. Google wallet, 2011. http://www.google.com/wallet/. Referenced November, 2011.

[79] G. Scott Graham and Peter J. Denning. Protection: principles and practice. In *Proceedings of the Spring Joint Computer Conference (SJCC)*, pages 417–429, May 1972.

[80] Trusted Computing Group. Subject key attestation evidence extension: version 1.0, June 2005. Available at: https://www.trustedcomputinggroup.org/.

[81] Trusted Computing Group. Mobile Trusted Module specification: version 1.0, June 2008. Available at: http://www.trustedcomputinggroup.org/.

[82] Trusted Computing Group. Mobile Trusted Module 2.0 use cases specification: version 1.0, March 2011. Available at: http://www.trustedcomputinggroup.org/.

[83] Trusted Computing Group. TPM main: Part 1 design principles: version 1.2, March 2011. Available at: http://www.trustedcomputinggroup.org/.

[84] 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. In *Proceedings of the USENIX Security Symposium*, pages 45–60, July 2008.

[85] J. Alex Halderman, Brent Waters, and Edward W. Felten. A convenient method for securely managing passwords. In *Proceesings of the International World Wide Web Conference (WWW)*, pages 471–479, May 2005.

[86] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Communications of the ACM*, 19:461–471, August 1976.

[87] Pieter H. Hartel and Luc Moreau. Formalizing the safety of Java, the Java virtual machine, and JavaCard. *ACM Computing Surveys*, 33(4):517–558, December 2001.

[88] Hewlett-Packard. OpenVMS guide to system security: OpenVMS version 8.4, June 2010. Available at: http://www.hp.com/.

[89] Lloyd Hilaiel. How BrowserID works, June 2010. http://lloyd.io/how-browserid-works. Referenced: January 2011.

[90] Advanced Micro Devices Inc. AMD64 virtualization codenamed "Pacifica" technology, secure virtual machine architecture reference manual, May 2005.

[91] RSA Security Inc. Solution brief: RSA SecurID two-factor authentication, 2010. Available at: http://www.rsa.com/.

[92] Trent Jaeger, Reiner Sailer, and Xiaolan Zhang. Analyzing integrity protection in the SELinux example policy. In *Proceedings of the USENIX Security Symposium*, August 2003.

[93] Paul Karger, David Toll, Elaine Palmer, Suzanne McIntosh, Samuel Weber, and Jonathan Edwards. Implementing a high-assurance smartcard OS. In *Proceeding of the Financial Cryptography and Data Security (FC)*, volume 6052, pages 51–65, January 2010.

[94] Chongkyung Kil, Emre Can Sezer, Ahmed M. Azab, Peng Ning, and Xiaolan Zhang. Remote attestation to dynamic system properties: Towards providing complete system integrity evidence. In *Proceedings of the Conference on Dependable Systems and Networks (DSN)*, pages 115–124, June 2009.

[95] Douglas Kilpatrick. Privman: A library for partitioning applications. In *Proceeding of USENIX Annual Technical Conference*, pages 273–284, June 2003.

[96] Ryan Kim. PayPal unveils NFC Android-to-Android payments, July 2011. http://gigaom.com/2011/07/13/paypal-unveils-nfc-android-to-android-payments/. Referenced: December 2011.

[97] Engin Kirda and Christopher Kruegel. Protecting users against phishing attacks. *The Computer Journal*, 49(5):554–561, September 2006.

[98] Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon

Winwood. seL4: formal verification of an operating-system kernel. *Communications of the ACM*, 53(6):107–115, 2010.

[99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Proceeding of the Advances in Cryptology (CRYPTO)*, pages 789–789, August 1999.

[100] René Korthaus, Ahmad-Reza Sadeghi, Christian Stüble, and Jing Zhan. A practical property-based bootstrap architecture. In *Proceedings of the ACM workshop on Scalable trusted computing (STC)*, pages 29–38, November 2009.

[101] A. Gunes Koru, Dongsong Zhang, Khaled El Emam, and Hongfang Liu. An investigation into the functional form of the size-defect relationship for software modules. *IEEE Transactions on Software Engineering*, 35(2):293 –304, March–April 2009.

[102] Ulrich Kuehn, Klaus Kursawe, Stefan Lucks, Ahmad-Reza Sadeghi, and Christian Stueble. Secure data management in trusted computing. In *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 324–338, August 2005.

[103] Mozilla Labs. BrowserID, 2011. https://browserid.org/. Referenced: January 2011.

[104] Butler W. Lampson. Protection. In *Proceedings of the Fifth Princeton Symposium on Informastion Sciences and Systems*, pages 437–443, March 1971. Reprinted in ACM Operating Systems Review, 8:1, pages 18–24, January 1974.

[105] Matthias Lange, Steffen Liebergeld, Adam Lackorzynski, Alexander Warg, and Michael Peter. L4Android: a generic operating system framework for secure smartphones. In *Proceedings of the ACM workshop on Security and Privacy in Smartphones and Mobile devices (SPSM)*, pages 39–50, October 2011.

[106] Ruby Lee et al. Architecture for protecting critical secrets in microprocessors. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 2–13, May 2005.

[107] Xavier Leroy. On-card bytecode verification for JavaCard. In *Proceedings of the International Conference on Research in Smart Cards: Smart Card Programming and Security (ESMART)*, pages 150–164, September 2001.

[108] Xavier Leroy. Bytecode verification on Java smart cards. *Software: Practice and Experience*, 32(4):319–340, 2002.

[109] Alana Libonati, Jonathan M. McCune, and Michael K. Reiter. Usability testing a malware-resistant input mechanism. In *Proceesings of the Network and Distributed Systems Security Symposium (NDSS)*, February 2011.

[110] MULTOS Limited. MULTOS – the high security smart card OS, September 2005. Avalable at: http://www.multos.com/.

[111] Peter A. Loscocco and Stephen D. Smalley. Meeting critical security objectives with Security-Enhanced Linux. In *Proceedings of the Ottawa Linux Symposium*, July 2001.

[112] Mohammad Mannan, Beom Heyn Kim, Afshar Ganjali, and David Lie. Unicorn: two-factor attestation for data security. In *Proceedings of the ACM conference on Computer and Communications Security (CCS)*, pages 17–28, October 2011.

[113] Claudio Marforio, Aurelien Francillon, and Srdjan Capkun. Application collusion attack on the permission-based security model and its implications for modern smartphone systems. Technical Report 724, ETH Zurich, April 2011.

[114] Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil Gligor, and Adrian Perrig. Trustvisor: Efficient TCB reduction and attestation. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 143–158, May 2010.

[115] Jonathan M. McCune, Bryan Parno, Adrian Perrig, Michael K. Reiter, and Arvind Seshadri. Minimal TCB code execution (extended abstract). In *Proceeding of the IEEE Symposium on Security and Privacy (SP)*, pages 267–272, May 2007.

[116] 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 ACM European Conference on Computer Systems (EuroSys)*, pages 315–328, March 2008.

[117] Jonathan M. McCune, Adrian Perrig, and Michael K. Reiter. Safe passage for passwords and other sensitive data. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)*, February 2009.

[118] Sun Microsystems. JavaCard 3 platform: White paper, August 2008. Available at: http://www.javacardforum.org/.

[119] Steven J. Murdoch, Saar Drimer, Ross Anderson, and Mike Bond. Chip and PIN is broken. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 433–446, May 2010.

[120] Mohammad Nauman, Sohail Khan, and Xinwen Zhang. Apex: extending android permission model and enforcement with user-defined runtime constraints. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 328–332, April 2010.

[121] Mohammad Nauman, Sohail Khan, Xinwen Zhang, and Jean-Pierre Seifert. Beyond kernel-level integrity measurement: Enabling remote attestation for the Android platform. In *Proceedings of the International Conference on Trust and Trustworthy Computing (TRUST)*, pages 1–15, 2010.

[122] George C. Necula. Proof-carrying code. In *Proceedings of the ACM Symposium on Principles of Programming Languages (POPL)*, pages 106–119, January 1997.

[123] NFCNews. MTA taps Nokia for NFC transit ticketing, October 2011. http://www.nfcnews.com/2011/10/27/mta-taps-nokia-for-nfc-transit-ticketing. Referenced: November, 2011.

[124] Nokia. Qt SDK, 2011. Available at: http://www.developer.nokia.com/. Referenced: January 2012.

[125] Bryan Parno. Bootstrapping trust in a "trusted" platform. In *Proceedings of the Conference on Hot Topics in Security (HOTSEC)*, July 2008.

[126] 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 IEEE Symposium on Security and Privacy (SP)*, pages 379–394, May 2011.

[127] Bryan Parno, Jonathan M. McCune, and Adrian Perrig. Bootstrapping trust in commodity computers. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 414–429, May 2010.

[128] Nathanael Paul and David Evans. .NET security: lessons learned and missed from Java. In *Proceedings of the Computer Security Applications Conference (ACSAC)*, pages 272–281, December 2004.

[129] Ugo Piazzalunga, Paolo Salvaneschi, and Paolo Coffetti. The usability of security devices. In Lorrie Faith Cranor and Simson Garfinkel, editors, *Security and Usability: Designing Secure Systems That People Can Use*, pages 221–242. O'Reilly, August 2005.

[130] Benjamin C. Pierce. Bounded quantification is undecidable. In *Proceedings of the ACM Symposium on Principles of Programming Languages (POPL)*, pages 305–315, January 1992.

[131] Jonathan Poritz, Matthias Schunter, Els Van Herreweghen, and Michael Waidner. Property attestation – scalable and privacy-friendly security assessment of peer computers. Technical Report RZ3548, IBM Research, October 2004.

[132] Jean-Jacques Quisquater and David Samyde. Electro magnetic analysis (EMA): Measures and counter-measures for smart cards. In *Proceedings of the International Conference on Research in Smart Cards: Smart Card Programming and Security (ESMART)*, pages 200–210, September 2001.

[133] Wolfgang Rankl and Wolfgang Effing. *Smart Card Handbook*, volume 3. Wiley, December 2003.

[134] Juniper Research. Mobile ticketing: Transport, sport, entertainment and events, October 2008. Available at: http://www.juniperresearch.com/.

[135] Blake Ross, Collin Jackson, Nicholas Miyake, Dan Boneh, and John C. Mitchell. Stronger password authentication using browser extensions. In *Proceedings of the USENIX Security Symposium*, July 2005.

[136] Ahmad-Reza Sadeghi and Christian Stüble. Property-based attestation for computing platforms: caring about properties, not mechanisms. In *Proceedings of the New Security Paradigms Workshop (NSPW)*, pages 67–77, September 2004.

[137] Ahmad-Reza Sadeghi, Marko Wolf, Christian Stüle, N. Asokan, and Jan-Erik Ekberg. Enabling fairer digital rights management with trusted computing. In *Proceeding of the International Conference on Information Security (ISC)*, pages 53–70, October 2007.

[138] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the USENIX Security Symposium*, August 2004.

[139] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

[140] Stuart E. Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. The emperor's new security indicators. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 51–65, May 2007.

[141] Dries Schellekens, Pim Tuyls, and Bart Preneel. Embedded trusted computing with authenticated non-volatile memory. In *Proceedings of the International conference on Trusted Computing and Trust in Information Technologies (TRUST)*, pages 60–74, March 2008.

[142] Gerhard Schellhorn, Wolfgang Reif, Axel Schairer, Paul A. Karger, Vernon Austel, and David Toll. Verification of a formal security model for multiapplicative smart cards. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, pages 17–36, October 2000.

[143] Jared Schmitz, Jason Loew, Jesse Elwell, Dmitry Ponomarev, and Nael Abu-Ghazaleh. TPM-SIM: a framework for performance evaluation of trusted platform modules. In *Proceedings of the Design Automation Conference (DAC)*, pages 236–241, June 2011.

[144] Michael D. Schroeder and Jerome H. Saltzer. A hardware architecture for implementing protection rings. *Communications of the ACM*, 15:157–170, March 1972.

[145] R. Sekar, V.N. Venkatakrishnan, Samik Basu, Sandeep Bhatkar, and Daniel C. DuVarney. Model-carrying code: a practical approach for safe execution of untrusted applications. In *Proceedings of the ACM symposium on Operating systems principles (SOSP)*, pages 15–28, October 2003.

[146] Marcel Selhorst, Christian Stüble, Florian Feldmann, and Utz Gnaida. Towards a trusted mobile desktop. In *Proceedings of the International conference on Trust and trustworthy computing (TRUST)*, pages 78–94, June 2010.

[147] Asaf Shabtai, Yuval Fledel, and Yuval Elovici. Securing Android-powered mobile devices using SELinux. *IEEE Security and Privacy*, 8(3):36–44, May 2010.

[148] Hovav Shacham. The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86). In *Proceedings of the ACM conference on Computer and Communications Security (CCS)*, pages 552–561, October 2007.

[149] Naeem Shaikh. Nokia money launched across india, December 2011. http://www.bgr.in/manufacturers/nokia/nokia-money-launched-across-india/. Referenced: December 2011.

[150] Aishvarya Sharma. On-board credentials: Hardware-assisted secure storage of credentials. Master's thesis, Helsinki University of Technology, January 2007.

[151] Steve Sheng, Bryant Magnien, Ponnurangam Kumaraguru, Alessandro Acquisti, Lorrie Faith Cranor, Jason Hong, and Elizabeth Nunge. Anti-phishing phil: the design and evaluation of a game that teaches people not to fall for phish. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*, pages 88–99, July 2007.

[152] Sean W. Smith. Outbound authentication for programmable secure coprocessors. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, pages 72–89, October 2002.

[153] Edward Suh, Charles O'Donnell, Ishan Sachdev, and Srinivas Devadas. Design and implementation of the AEGIS single-chip secure processor using physical random function. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, pages 25–36, June 2005.

[154] Sandeep Tamrakar, Jan-Erik Ekberg, and N. Asokan. Identity verification schemes for public transport ticketing with NFC phones. In *Proceedings of the ACM workshop on Scalable Trusted Computing (STC)*, pages 37–48, October 2011.

[155] Sandeep Tamrakar, Jan-Erik Ekberg, Pekka Laitinen, N. Asokan, and Tuomas Aura. Can hand-held computers still be better smart cards? In *Proceedings of the Trusted Systems International Conference (INTRUST)*, pages 200–218, December 2010.

[156] David C. Toll, Paul A. Karger, Elaine R. Palmer, Suzanne K. McIntosh, and Sam Weber. The Caernarvon secure embedded operating system. *ACM Operating Systems Review*, 42:32–39, January 2008.

[157] Marten van Dijk, Jonathan Rhodes, Luis F. G. Sarmenta, and Srinivas Devadas. Offline untrusted storage with immediate detection of forking

and replay attacks. In *Proceedings of the ACM workshop on Scalable Trusted Computing (STC)*, pages 41–48, November 2007.

[158] Amit Vasudevan, Bryan Parno, Ning Qu, and Adrian Perrig. Lockdown: A safe and practical environment for security applications. Technical Report CMU-CyLab-09-011, Carnegie Mellon University, July 2009.

[159] Luca Viganò. Automated security protocol analysis with the AVISPA tool. *Electronic Notes in Theoretical Computer Science*, 155:61–86, May 2006.

[160] Marcia Villalba. One-time passwords and remote credential management using On-board Credentials. Master's thesis, Aalto University, February 2011.

[161] Dan S. Wallach, Dirk Balfanz, Drew Dean, and Edward W. Felten. Extensible security architectures for Java. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 116–128, October 1997.

[162] Thomas Weigold, Thorsten Kramp, Reto Hermann, Frank Hring, Peter Buhler, and Michael Baentsch. The Zurich trusted information channel – an efficient defence agains man-in-the-middle and malicious software attacks. In *Proceedings of the International Conference on Trusted Computing and Trust in Information Technologies (TRUST)*, pages 75–91, March 2008.

[163] Maurice V. Wilkes. *The Cambridge CAP computer and its operating system (Operating and programming systems series)*. North-Holland Publishing Co., September 1979.

[164] Johannes Winter. Trusted computing building blocks for embedded Linux-based ARM TrustZone platforms. In *Proceedings of the ACM workshop on Scalable Trusted Computing (STC)*, pages 21–30, October 2008.

[165] Min Wu, Robert C. Miller, and Simson L. Garfinkel. Do security toolbars actually prevent phishing attacks? In *Proceedings of the Conference on Human factors in computing systems (CHI)*, pages 601–610, April 2006.

[166] Jeff Yan, Alan Blackwell, Ross Anderson, and Alasdair Grant. Password memorability and security: Empirical results. *IEEE Security and Privacy*, 2(5):25–31, October 2004.

[167] Dino A. Dai Zovi. Apple iOS 4 security evaluation. In *Black Hat USA*, July 2011.

Traditional hardware credentials are expensive for service providers and often inconvenient for end-users. Software-based credentials may offer better usability at low cost but such solutions are vulnerable to many common attacks. In this dissertation, we present On-board Credentials (ObC), a novel credential platform for mobile devices. The ObC platform enables development of credentials that are simultaneously secure, user-friendly and inexpensive. The distinguishing feature of the ObC platform is an open provisioning model that allows any service provider to develop and deploy new credential types freely. We have implemented the ObC platform for existing mobile devices. The first substantial credential deployments are currently starting.

Aalto-DD 49/2012

ISBN 978-952-60-4597-9
ISBN 978-952-60-4598-6 (pdf)
ISSN-L 1799-4934
ISSN 1799-4934
ISSN 1799-4942 (pdf)

Aalto University
School of Science
Department of Computer Science and Engineering
www.aalto.fi