Department of Computer Science

Applications of Trusted Execution Environments (TEEs)

Sandeep Tamrakar

Aalto University

DOCTORAL DISSERTATIONS

Aalto University publication series
DOCTORAL DISSERTATIONS 105/2017

Applications of Trusted Execution Environments (TEEs)

Sandeep Tamrakar

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Science, at a public examination held at the lecture hall AS1 of the school on 19th June 2017 at 12 noon.

Aalto University
School of Science
Department of Computer Science
Secure Systems

Aalto University publication series
DOCTORAL DISSERTATIONS 105/2017

© Sandeep Tamrakar

ISBN 978-952-60-7464-1 (printed)
ISBN 978-952-60-7463-4 (pdf)
ISSN-L 1799-4934
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)
http://urn.fi/URN:ISBN:978-952-60-7463-4

Unigrafia Oy
Helsinki 2017

Finland

NORDIC ECOLABEL
441 697
Printed matter

Aalto University

Abstract

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

Author	Sandeep Tamrakar
Name of the doctoral dissertation	Applications of Trusted Execution Environments (TEEs)
Publisher	School of Science
Unit	Department of Computer Science
Series	Aalto University publication series DOCTORAL DISSERTATIONS 105/2017
Field of research	Computer science, security
Manuscript submitted	13 March 2017
Date of the defence	19 June 2017
Permission to publish granted (date)	8 May 2017
Language	English

☐ **Monograph** ☒ **Article dissertation** ☐ **Essay dissertation**

Abstract

Trust is vital for arbitrary entities to interact and cooperate. These entities may have different security requirements. Trust allows them to ensure that they will behave correctly and fulfill each other's security requirements as well as assure their privacy. A Trusted Execution Environment (TEE) is one available technology that can be used to establish trust between entities. TEEs are widely deployed on device platforms, and recently they have also begun to appear on server platforms.

In multilateral scenarios, hardware-based TEEs allow us to build efficient protocols and systems for ensuring security requirements of the non-trusting entities and assuring their privacy. In this dissertation, I consider two separate use cases where trust is required at the user's end: hosting credentials such as electronic identity on users' devices (e.g. mobile phones), and using NFC-enabled devices for hosting public transport ticketing credentials. I present a TEE-based architecture for hosting different types of credentials securely on users' devices, and using them from the devices over various communication channels (e.g. USB and NFC). I also show how to use TEEs to assure user-to-device binding, and attest the level of security on devices for remote credential provisioning. These solutions are supported by implementations on real mobile devices with hardware TEEs based on ARM TrustZone. I also show an example of how to use TEEs to ensure users' data privacy while accessing services on third-party infrastructure. For this, I consider the use case of cloud-based mobile malware checking where users submit queries about their mobile applications to an untrusted server, which processes users' queries in a TEE and returns the results without learning anything about the content of the queries. A prototype of this service was built using two different hardware TEE platforms: ARM TrustZone and Intel SGX.

The work described in this dissertation takes advantage of the programmability offered by TEEs to implement application-specific security functionality. However, other non-programmable trusted hardware, such as TPMs, can also be used as trust anchors. I compare and contrast programmable versus non-programmable trusted hardware, considering the functionality and interfaces each offers. Further, I present a categorization of credentials based on their migration policies and discuss possible mechanisms to migrate/share credentials among other devices belonging to the same users. I also discuss the importance of a trusted path for user-to-TEE interactions and present an overview of the currently available mechanisms to establish a trusted path. Finally, I describe how to leverage a combination of TEEs on users' devices as well as the infrastructure to enhance the security of applications and further develop new types of services.

Keywords	Trusted Execution Environment, Security				
ISBN (printed)	978-952-60-7464-1	**ISBN (pdf)**	978-952-60-7463-4		
ISSN-L	1799-4934	**ISSN (printed)**	1799-4934	**ISSN (pdf)**	1799-4942
Location of publisher	Helsinki	**Location of printing**	Helsinki	**Year**	2017
Pages	216	**urn**	http://urn.fi/URN:ISBN:978-952-60-7463-4		

Preface

This dissertation is a result of my work and learning from my internship at Nokia Research Center, Finland and my doctoral studies at the Department of Computer Science in Aalto University School of Science.

I am more than grateful to my supervisor Professor N. Asokan for his excellent guidance, encouragement, support, and patience. It is my privilege to have you as my supervisor. Dr. Jan-Erik Ekberg has been the biggest stimulating instructor for my dissertation. Work that constitutes this dissertation would never have excelled without his invaluable stream of advice, ideas, and experiences. I would like to extend my gratitude to my instructor Dr. Andrew Paverd, for his excellent guidance and enthusiastic engagement towards the successful completion of this dissertation.

I am also grateful to Professor Tuomas Aura for his support at various stages of my doctoral studies. I would like to thank all my co-authors that have contributed to this work. Special thanks to Pekka Laitinen and Jian Liu for all the insightful discussion during my research. Thanks to my colleagues Dr. Kari Kostiainen, Dr. Markku Antikainen, Thomas Nyman, Thanh Bui, Siddharth Rao, and Manish Thapa with whom I share an enjoyable working experience. I am indebted to my pre-examiners Professor Dr. René Mayrhofer and Prof. Konstantinos Markantonakis for their thorough reviews and insightful feedback. Additionally, I would like to thank Professor Ville Leppänen for the honor of having him as the opponent for my dissertation.

My research was supported by TEKES funded projects Flexible Services program and Cloud-assisted Security Services (CloSer), and Academy of Finland project Cloud Security Services (CloSe). Additionally, Nokia Foundation has also generously supported my research.

I would also like to thank my friends Gautam Raj Moktan, Prem Raj Adhikari, Prajwal Osti, Subash Khanal and Manik Madhikermi who made my life in Finland wonderful. I express my sincere gratitude to my parents and my family for their everlasting encouragement, blessings, and love. Finally, it is my privilege to thank my soulmate Anjala who endure all the boredom during my work. Thank you for your support, motivation and unconditional love.

Espoo, May 16, 2017,

Sandeep Tamrakar

Contents

List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I Sandeep Tamrakar, Jan-Erik Ekberg, Pekka Laitinen, N. Asokan and Tuomas Aura. Can Hand-Held Computers Still Be Better Smart Cards?. In *International Conference on Trusted Systems (InTrust 2010)*, pages 200 – 218, December 2010.

II Sandeep Tamrakar, Jan-Erik Ekberg, and N. Asokan. Identity Verification Schemes for Public Transport Ticketing with NFC Phones. In *Proceedings of the Sixth ACM Workshop on Scalable Trusted Computing*, Chicago, Illinois, USA, pages 37 – 48, October 2011.

III Jan-Erik Ekberg and Sandeep Tamrakar. Mass Transit Ticketing with NFC Mobile Phones. In *International Conference Trusted Systems (InTrust 2011)*, Beijing, China, pages 48 – 65, November 2011.

IV Sandeep Tamrakar and Jan-Erik Ekberg. Tapping and Tripping with NFC. In *International Conference on Trust and Trustworthy Computing (TRUST 2013)*, **Won the Best Paper Award**, London, UK, pages 115 – 132, June 2013.

V Sandeep Tamrakar, Jan-Erik Ekberg, Pekka Laitinen. On Rehoming the Electronic ID to TEEs. In *IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom-15)*, Helsinki, Finland, pages 49 – 56, Volume:1, August 2015.

VI Sandeep Tamrakar, Jian Liu, Andrew Paverd, Jan-Erik Ekberg, Benny Pinkas, and N. Asokan. The Circle Game: Scalable Private Membership Test Using Trusted Hardware. In *ACM Asia Conference on Computer and Communications Security (ASIACCS) 2017*, **Honorable Mention**, Abu Dhabi, UAE, pages 31 – 44, April 2017.

Other Publications

The following publications are not included in this dissertation. Publication X is referred to in this dissertation.

VII Alexandra Dmitrienko, Ahmad-Reza Sadeghi, Sandeep Tamrakar and Christian Wachsmann. SmartTokens: Delegable Access Control with NFC-Enabled Smartphones. In *International Conference on Trust and Trustworthy Computing (TRUST 2012)*, pages 219–238, June, 2012.

VIII Babins Shrestha, Manar Mohamed, Anders Borg, Nitesh Saxena, and Sandeep Tamrakar. Curbing mobile malware based on user-transparent hand movements. In *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 219–238, March, 2015.

IX Babins Shrestha, Manar Mohamed, Sandeep Tamrakar and Nitesh Saxena. Theft-resilient mobile wallets: transparently authenticating NFC users with tapping gesture biometrics. In *Proceedings of the thirty-second Annual Conference on Computer Security Applications (ACSAC)*, pages 265–276, December, 2016.

X Andrew Paverd, Sandeep Tamrakar, Hoang Long Nguyen, Praveen Kumar Pendyala, Thien Duc Nguyen, Elizabeth Stobert, Tommi Gröndahl, N. Asokan, Ahmad-Reza Sadeghi. OmniShare: Securely Accessing Encrypted Cloud Storage from Multiple Authorized Devices. *IEEE Internet Computing* (to appear), 2017.

List of Abbreviations

AIK	Attestation Identity Key
API	Application Programming Interfaces
CA	Client Application
CoaC	Cuckoo-on-a-Carousel
CoO	Cuckoo-on-ORAM
CSR	Certificate Signing Request
DAA	Direct Anonymous Attestation
DRAM	Dynamic RAM
DRM	Digital Rigths Management
EA	Enhanced Authorization
ECC	Elliptical Curve Cryptography
eID	electronic Identity
EK	Endorsement Key
eMMC	Embedded Multimedia Card
EPID	Enhanced Privacy ID
EPS	Endorsement Primary Seed
ESP	Electronic Signature Products
EU	European Union
FINeID	Finnish Electronic Identity
FPR	False Positive Rate
GSMA	Global System for Mobile Association
IEC	International Electrotechnical Commission
IFD	Interface Device
ISO	International Standardization Organisation
JCA	Java Cryptography Architecture
kB	Kilobyte

LLCP	Logical Link Control Protocol
MAC	Message Authentication Code
MB	Megabyte
MMU	Memory Management Unit
MNO	Mobile Network Operator
ms	millisecond
NDEF	NFC Data Exchange Formats
NFC	Near Field Communication
NS	Non-Secure
OAEP	Optimal Asymmetric Encryption Padding
ObC	On-board Credentials
OEM	Original Equipment Manufacturer
OP-TEE	Open Portable Trusted Execution Environment
OPK	ObC Platform Key
ORAM	Oblivious RAM
OS	Operating System
OTI	Open Ticketing Institute
OTrP	Open Trust protocol
PC	Personal Computer
PCA	Privacy Certificate Authority
PC/SC	Personal Computer/Smart Card
PCD	Proximity Coupling Device
PCR	Platform Configuration Registers
PDA	Personal Digital Assistant
PICC	Proximity Integrated Circuit Cards
PMT	Private Membership Test
PPS	Platform Primary Seed
PSI	Private Set Intersection
RA	Registration Authority
RAM	Random Access Memory
REE	Rich Execution Environment
RPMB	Replay Protected Memory Block
SCE	Smart Card Engine
SCS	Signature Creation Service
SE	Secure Element
SGX	Software Guard Extensions

SIM Subscriber Identity Module

SKAE Subject Key Attestation Evidence

SMC Secure Monitor Call

SoC System on a Chip

SPS Storage Primary Seed

SRK Storage Root Key

TA Trusted Application

TCB Trusted Computing Base

TCG Trusted Computing Group

TEE Trusted Execution Environment

TLB Translation Look-aside Buffer

TPM Trusted Platform Module

TSM Trusted Service Manager

UCOM User Centric Owner Model

UI User Interface

UICC Universal Integrated Circuit Card

UUID Universally Unique Identifier

VM Virtual Machine

VSC Virtual Smart Card

vTPM virtual TPM

1. Introduction

In multilateral environments, hardware-based trusted execution environments (TEEs) can be used to build protocols for ensuring security requirements of the different (non-trusting) entities and assuring their privacy. TEE-based protocols can provide better security guarantees or lower communication overhead compared to alternative approaches. In this dissertation, I show how to use TEEs to build protocols and systems for provisioning credentials on users' devices remotely, protect them, enforce policies on their use, and use them securely. I also show an example of how to use TEEs to ensure users' privacy while accessing services on untrusted infrastructure. In this dissertation, I used the term *credential* to refer to data objects that an entity (user/device) can use to demonstrate its right to use/access services offered by other entities. Parts of a credential can be public information such as public key certificates and others can be confidential information, such as secret keys.

1.1 Motivation

Most online applications and services that require user identification are *multilateral* by nature i.e. they involve multiple entities. For example, a user may access e-health services from her web browser using a government-issued electronic identity (eID). In this example, the entities involved are (at minimum) the user, the e-health service, and the identity provider. Different entities may have different security requirements. However, these entities do not necessarily trust each other [81].

Defining security associations involving the credentials of each entity allows the non-trusting entities to communicate with each other securely.

Credentials allow entities to authenticate themselves while accessing services offered by others entities. Enforcing policies on the use of credentials allows establishing trust between entities. For example, a service provider may only allow service access to users with credentials issued by a certain credential issuer. Additionally, the credential issuer may also require users to store credentials on devices, such as smart cards, that ensure the integrity and confidentiality of the credentials, and protect them from unauthorized access.

Software-based solutions: Existing software-based solutions alone are not sufficient to protect these credentials and enforce policies. For example, a private key can be locally stored on a user's device (e.g., on a device's local storage). The key can also be transferred or shared among other devices as well as with other users. Thus, they do not guarantee the authenticity of users. Many services require the identity verification of users before allowing service access. In services such as digital rights management (DRM), providers bind digital content to specific credentials and restrict the duplication of credentials [87]. Therefore, dedicated, tamper-resistant hardware, such as smart cards or Trusted Platform Modules (TPMs) that restrict credential duplication are needed for storing DRM credentials. Furthermore, TPMs also allow providers to enforce policies that prevent the content being accessed on other devices, even those owned by the same user [83, 69].

Smartcard-based solutions: In addition to preventing credential duplication, dedicated hardware such as smart cards are ideal for accessing services where card readers are part of the service infrastructure. For example, in public transport systems, station gates are equipped with contactless card readers that can verify users' ticketing credentials stored on their contactless cards (e.g. travel cards) and allow users to enter/exit the gates. However, not all services provide such card readers. In some cases, the service providers expect users to arrange a card reader for themselves. For example, accessing an e-health service online with eID cards requires users to purchase a card reader and attach it to their personal computers (PCs).

A major drawback of smartcard-based credentials is the cost of deployment; Service providers must provision credentials on smart cards and physically distribute them to users. Also, a smartcard-based credential often serves a single purpose. Therefore, a user subscribed to multiple services operated by separate providers ends up with many cards that need to be carried everywhere for service use. Further, maintaining and updating smartcard-based credentials after they are issued is often difficult. Often existing cards are replaced with new cards while updating services. For example, the Helsinki Regional Transport Authority (HSL) has announced that they will replace current travel cards to support online loading of the ticket values on cards.[1]

Trust assurances: Users may also desire certain trust assurances, such as data security and privacy guarantees, from the service infrastructure. For example, in an electronic voting system, citizens may require the following assurances: the integrity of their votes, the confidentiality of their voting choices and the protection of their identities.

Thus, establishing trust in a multilateral environment requires mechanisms that:

(a) facilitate the secure hosting of different types of credentials on users' devices and enforce security policies on their use, and

(b) ensure the integrity and confidentiality of users' data processed at the service infrastructure as well as provide the necessary privacy guarantees.

Trusted Execution Environments: A *Trusted Execution Environment* (TEE) is one available technology that can be used to establish trust between entities. It provides hardware-assisted isolation of code and data from all other software running on the platform including the operating system (OS). Some TEEs also support remote attestation that guarantees to third parties that specific code is running in the TEE. Thus, it can be used to run application-specific security logic for hosting credentials and enforcing security policies on them. This dissertation shows how to use a TEE as a trust anchor to develop applications and services in multilateral environments. Specifically, I present how to use TEEs to establish trust in users' devices to allow remote provisioning and secure the usage of credentials, and how to use TEEs on service infrastructure to ensure and assure privacy while handling users' data.

[1]HSL travel card change: `https://www.hsl.fi/en/news/2015/travel-cards-changed-blue-hsl-cards-2016-6184`

1.1.1 Establishing trust on users' devices

Devices such as smartphones and tablets, have become de facto personal computing devices, which are always within the vicinity of their users. Therefore they are an ideal place to host credentials. They also offer a wide range of built-in features such as sensing and communication capabilities, user input/output interfaces, and most importantly they provide programmability.

Although OSs for smartphones are designed to be more secure than OSs for PCs [60, 99] their security is routinely defeated using techniques such as *jailbreaking* on iOS and *rooting* on Android OS [99]. To host a credential on a device, we must ensure that the device protects against unauthorized use of the credential, e.g. by malware. Additionally, the device must also be able to protect credentials from misbehaving users. For example, a user should not be allowed to modify the validity period of an expired credential. Since credential issuers do not have control over users' devices, they need assurances from a device prior to credential provisioning so that, a) the device complies with their security requirements to protect the credentials, and b) it belongs to the legitimate user to whom the credentials are being issued.

Smartcard-based solutions: One possibility is to use smartcard-based solutions such as a subscriber identity module (SIM) available on devices. For example, a mobile certificate is a user-specific credential hosted on the user's SIM. The mobile network operators (MNOs) control access to SIMs. They can provision and update applications on SIMs remotely. Further, if a user is a registered customer of an MNO, the MNO can verify the identity of the SIM user while provisioning credentials remotely. However, provisioning credentials on SIMs from other service providers requires permission from the MNOs. Also, it is not likely that all services that a user intends to access would accept credentials from a single credential issuer. Thus, to reduce such complexity, the Global System for Mobile Association (GSMA) has introduced a trusted service manager (TSM) [22], which acts as a single trusted entity responsible for distribution and service management on smart cards such as SIMs. However, service providers are required to make agreements with a TSM which usually involves service fees. Further, making service agreements with TSMs in multiple geographical locations can become infeasible for smaller service providers. To overcome these complexities, a *User-centric* card ownership models such as the *user centric owner model* (UCOM) [1] have been proposed, which allows users to host applications of their choice on their cards without permission from the card issuer. (Section 2.1 briefly describes the smart card technology and user-centric card ownership models.)

Trusted hardware-based solutions: Modern device OS platform such as iOS have introduced trusted hardware-backed payment services called Apple Pay.[2] This service allows users to integrate their card-based payments (e.g. debit, credit, and loyalty) functionality into the payment service running on their device. Users receive device-specific virtual account numbers for each payment card that they add to service. The device protects the virtual account information using a trusted hardware called Secure Element (SE). During payment at a payment terminal, the SE provides the virtual account number along with a transaction-specific dynamic security code to the terminal. Later, bank approves the payment after verifying the account number and the security code. Android has also released similar service called Android pay.[3] However, it relies on cloud service to generate transaction tokens. These services are only limited to payments and require agreements between the service providers and the payment card operators (e.g. credit card companies or banks).

Recently, Android and iOS have supported a hardware-backed keystore to protect cryptographic keys and operations [40, 4]. However, hardware-backed keystores do not allow hosting arbitrary credential algorithms. Alternatively, TEEs, are widely available on modern smartphone platforms [21] Various researchers [21, 99] have proposed using a TEE as a trust anchor for hosting credentials on user's devices. Particularly, Kostiainen in his dissertation [56] presented an On-board Credential architecture [59] for building an open credential platform that allows service providers to provision applications and credentials to a TEE without requiring permission from the TEE manufacturer. (ObC was available on devices such as Nokia Windows Phone 8 and Symbian phones.) However, questions remain on how to associate users' identities with their devices

[2]`https://support.apple.com/en-gb/HT203027`

[3]`https://www.android.com/pay/`

for remote credential provisioning, and how to manage and use different types of credentials on a single device.

In Chapter 3, this dissertation shows how to use a TEE as a trust anchor for hosting and provisioning credentials. It also presents an example of using credentials hosted on a device to access online services from an external device such as a PC over a USB channel. In Chapter 4, it presents the use of credentials stored on devices over low bandwidth channels such as Near Field Communication (NFC). Particularly, it presents various identity verification schemes that can be used in public transport ticketing from NFC-enabled phones.

1.1.2 Establishing trust on infrastructure

Service providers set out their intentions for using user data in their *terms and conditions* statements to which users must agree to when signing up for services. Service providers usually comply with the local data protection laws where their servers are hosted. For example in the European Union (EU), the Data Protection Directive (95/46/EC) [17] requires that anyone receiving personal data from the EU must comply with the EU data protection rules.[4]

Traditionally, computing infrastructure was maintained by the service providers, and they can conduct security audits to comply with the data protection laws. However, in recent years, cloud computing has gained significant attention from private individuals as well as commercial service providers. Cloud computing allows its consumers to outsource their computing needs to cloud providers, which operate and maintain the computing infrastructure. This includes servers and networks that can be consumed as a resource in a scalable and flexible manner on demand. Adopting cloud computing technologies reduces the cost of maintaining and upgrading the computing infrastructure for service providers, and allows them to focus more on service development. As a result, many local and remote services have migrated to cloud platforms (e.g. cloud-assisted malware checking services such as the *Verify Apps* feature on Android [41]).

[4]The General Data Protection Regulation (2016/679) will replace the Data Protection Directive 95/46/EC from May 2018.

Cloud-assisted malware checking: Malware checking services used to operate on users' devices where a locally-installed anti-malware tool regularly received lists of known malware identifiers and checked the device for malware. In a cloud-assisted design, anti-malware vendors provide malware databases to the cloud where they host malware checking services. Then users can send application identifiers to the service and receive the status of the corresponding applications as responses. The cloud-assisted design is beneficial to anti-malware vendors as it allows them to serve all users with the latest malware database, and most importantly it allows anti-malware vendors to retain their malware databases as a potential competitive advantage without having to disclose the databases to users (and thus competitors).

Privacy concerns: However, *data privacy* is a major concern in cloud platforms [24]. As computation is outsourced to third-party cloud servers, the clients of the cloud providers (e.g. the service providers) no longer retain their control over their users' data. A malicious cloud provider can easily access and observe their clients' data to profile users. Another concern arises from the multi-tenancy architecture of the cloud that allows computing resources to be shared among multiple clients for maximum resource utilization. Cloud providers use virtualized environments to achieve multi-tenancy, where they host applications and data belonging to different clients on the same physical server. Vulnerabilities in the virtualization techniques can lead to cross-virtual machine information leakage that allows a malicious application on a guest virtual machine (VM) to collect data from other co-located VMs [75]. Service providers may choose to encrypt their users' data before storing it in the cloud, but this precludes outsourcing any processing to the cloud. Also, the adversary can still observe data access patterns to gather information.

Therefore, cloud-assisted services such as malware checking should protect against the adversary learning users' query contents which can be used to profile users. One way to provide query privacy is to run the malware checking service as a *private membership test* (PMT). In PMT, a *lookup server* holds a large set of entries called a *dictionary* X. A user runs a PMT protocol with the lookup server to know if an entry q is a member of X. The protocol outputs a single bit r indicating the membership status of q without revealing q and r to the lookup server.

Trust assurances: A survey by Ardagna et al. [5] calls for an increase in cloud transparency through the use of standardized interfaces to improve the trustworthiness of the cloud. One method of building trust assurance for services like PMT is to use a trusted platform module (TPM) [95]. A TPM allows the attestation of the state of the platform to third parties. TPM attestation can be used to provide assurance that a service running on the platform is operating in a correct fashion (service integrity) and to assure that no malicious activities are running on the platform. However, attesting individual applications to third parties using a TPM is challenging. Also, facilitating TPM attestation in a virtualized environment is still an open question. Cryptographic solutions such as private set intersection [77, 78] and homomorphic encryption [68] have been proposed for privacy-preserving membership tests. However, they are expensive in terms of computation and communication when supporting multiple clients.

In recent years, TEEs have started to appear for server platforms — Intel's Software Guard Extensions (SGX) [67] will soon be available for cloud platforms. SGX provides attestation in addition to runtime code isolation, which can be used to design privacy-preserving services such as PMT. Several researchers [84, 9] have used TEEs to build *trusted cloud computing* platforms that provide confidentiality and integrity of applications and their data on cloud platforms. In Chapter 5, this dissertation shows how to utilize TEEs to build privacy-preserving services, such as PMT, on untrusted infrastructure.

1.2 Contributions

The contributions of the individual publications that constitute this dissertation are summarized below:

Publication I: "Can Hand-Held Computers Still Be Better Smart Cards?"

Research contributions: Virtual smart cards (VSCs) have been implemented using hand-held devices for more than a decade. Balfanz et al. [8] presented a VSC architecture where the smart card functionality was implemented on a Personal Digital Assistant (PDA) rather than as a separate hardware module. Publication I presents a revised architecture for VSC with hardware support.

It uses a TEE available on a user's device to protect security-sensitive smartcard operations from the influence of other applications including the OS running on the device. The hardware-assisted VSC allows hosting and managing different types of smartcard-based credentials on a single device. It presents a reference implementation of a real smartcard-based credential, Finnish electronic identity (FINeID), as a hardware-supported VSC instance. Additionally, the user's device emulates USB interfaces similar to a standard USB card reader, which allow external devices, such as PCs, to access the hardware-supported VSCs in a similar manner as they access physical smart cards via USB-connected card readers.

Author's contributions: I contributed to the design and analysis of the hardware-supported VSC architecture. I also implemented the Linux version of the hardware-supported VSC architecture and the emulation of USB card reader interfaces on a Nokia 900 device running Maemo 5. In additional, I collected and analyzed the performance measurements from this platform. Pekka Laitinen provided the implementation and performance measurements on Nokia X6 running Symbian OS. I wrote the manuscript together with my co-authors.

Publication II: "Identity Verification Schemes for Public Transport Ticketing with NFC Phones"

Research contributions: Public transport systems around the world use contactless cards as transport tickets. Ticketing applications on cards are usually proprietary and are incompatible between different transport systems. Account-based ticketing is an alternative approach to overcome this incompatibility and allow flexible ticket fare collection schemes [88]. In account-based ticketing, each traveler is represented by a travel account maintained by an identity provider. While traveling, the checkpoint terminals at transport endpoints (entries and exits) verify travelers' identities before allowing them to travel. During identity verification, the checkpoint terminals collect travelers' identities and bind them with contextual information (e.g. date, time, location information) to produce *ticketing evidence*.

Publication II presents four protocol variants of identity verification that can be used in an account-based ticketing architecture with NFC-enabled devices, particularly NFC-enabled mobile phones. The architecture uses TEEs on NFC-enabled devices to protect users' ticketing credentials as well as to perform cryptographic operations using these credentials.

The publication identifies that communication speed, rather than cryptographic computation, is the main bottleneck for implementing identity verification schemes on NFC-enabled mobile phones. The actual data rate between NFC applications on mobile phones is around 1000 bytes per second. At this rate, it would require up to a second to transfer an X.509 certificate using a standards-compliant RSA digital signature with a key size of 1024 bits. However, it is expected that user interaction in public transport ticketing should conclude within 300 milliseconds (ms) [2]. Therefore, it is evident that a standards-compliant RSA digital signature scheme is impractical to use as an identity-verification scheme due to its message exchange size.

The proposed protocol variants are optimized in message sizes with minimum number of messages exchange during identity verification. It also studies the effects of optimization on the usability and security aspect of the system.

Author's contributions: I contributed to the design and analysis of the account-based ticketing architecture and the protocols for identity verification. I implemented the protocols on NFC-enabled phones, and I collected and analyzed the performance measurements. I wrote the manuscript together with my co-authors.

Publication III: "Mass Transit Ticketing with NFC Mobile Phones"

Research contributions: Publication III builds on Publication II to design an identity verification scheme that uses the RSA signature with message recovery to minimize the size of the public key certificate. Publication III also extends the account-based ticketing architecture to support *non-gated* transport systems, where the transport end-points are not controlled via physical gates.

In non-gated transport systems, offline resource-constrained devices, such as contactless smart cards, are placed at transport stations as checkpoint terminals. Travelers are required to tap-in/tap-out with their NFC-enabled devices at these terminals to produce ticketing evidence.

The architecture utilizes TEEs on NFC-enabled devices (NFC-enabled mobile phones and smart cards) to produce and protect ticketing evidence. Further, the TEE on a user's device also limits the number of times the device is allowed to perform identity verification before it must submit the ticketing evidence to a centralized authority (e.g. the identity provider).
Author's contributions: I contributed to the implementation of the ticketing application for the NFC-enabled phone, and the ticket reader application for the checkpoint terminals. Jan-Erik Ekberg designed the protocols as well as the implementation of the trusted application for the TEEs on the phones and the smart cards used as checkpoint terminals. I ran the experiments, collected data, and analyzed and evaluated the results of performance measurements. I wrote the manuscript and analyzed the implementation of the protocols together with Jan-Erik Ekberg.

Publication IV: "Tapping and Tripping with NFC"

Research contributions: Publication IV is an adaptation of the non-gated transport ticketing protocol presented in Publication III to a real world trial design. A month-long trial was conducted with over 100 participants along the Port Washington Branch of the Long Island Railway Road[5], operated by the New York Metropolitan Transit Authority.

The data collected during the trial showed almost 67% of the ticketing evidence was reported immediately and 80% on the same day. Such insights are useful to monitor the transport system utilization and predict congestion in near real-time. A user study conducted at the end of the trial showed that 42% of the participants preferred to use NFC-enabled phones for ticketing compared to only 24% who preferred travel cards.

Publication IV also presents an upgraded identity verification protocol designed for such NFC-enabled phones where a TEE is unavailable or practically inaccessible (open-devices). The protocol requires travelers to fetch access tokens directly from a centralized authority before identity verification. It also requires travelers to submit ticketing evidence before they are allowed to fetch new tokens.

[5] `http://www.mta.info/lirr`

Author's contributions: I contributed to the design and preparation of the trial. Furthermore, I adapted the protocol from Publication III into the trial settings and implemented the protocols as an extension to the Nokia Public Transport app used in the trial. I contributed to the design and implementation of the extended identity verification protocol to support open-devices. I also analyzed the data collected during the trial. The manuscript was written together with Jan-Erik Ekberg.

Publication V: "On Rehoming the Electronic ID to TEEs"

Research contributions: Publication V presents a system (software and a network architecture) that allows credential provisioning on hardware-supported VSCs remotely. It uses two separate device platforms (Android and Windows Phone), with different TEE providers (Kinibi and On-board Credentials respectively) to prototype remote provisioning of FINeID credentials. It utilizes remote attestation mechanisms defined by the corresponding TEE providers to assure the credential issuers of the level of security on the devices during credential provisioning. It also demonstrates an example of binding users' identities with their devices using a pre-existing strong digital identity, such as government-issued electronic identity (eID) cards.
Author's contributions: I contributed to defining the remote provisioning architecture and reviewing the state-of-the-art in eID systems. I also defined the security requirements for provisioning as well as using eIDs on TEEs, analyzed the protocols and wrote the manuscript together with my co-authors.

Publication VI: "The Circle Game: Scalable Private Membership Test Using Trusted Hardware"

Research contributions: Publication VI models the cloud-assisted malware checking scenario as a *private membership test* (PMT) and designs a PMT protocol that utilizes TEEs available on the server to provide query privacy. It introduces a simple PMT approach using a *carousel* design pattern where the entire malware dictionary is circled through trusted hardware on the cloud server. Publication VI also shows the carousel approach using different data structures to represent the dictionary, and implements a prototype on two different TEE architectures (ARM TrustZone and Intel SGX). Furthermore, through extensive experimental analysis, Publication VI shows that the carousel approach exhibits better scalability than ORAM while supporting a higher number of simultaneous queries.

Author's contributions: I conceived the idea of the carousel approach together with Jan-Erik Ekberg. The authors jointly contributed to the design specifically in selecting and comparing different data structures used for representing the dictionary. I implemented, measured, analyzed and evaluated the performance of the carousel approach on the ARM-based TEE platform. Andrew Paverd performed the implementation, measurements, and evaluation on the Intel SGX. I wrote the manuscript together with my co-authors.

1.3 Outline

This dissertation is based on the aforementioned six original publications which are grouped into three main themes. Chapter 2 presents the relevant background for the whole dissertation.

Chapter 3 encompasses Publication I and Publication V. It presents a hardware-supported virtual smart card architecture for hosting credentials on a single device. It shows how that architecture is better than dedicated hardware tokens or smart cards in terms of credential management and credential use. It also presents how to use TEEs to attest the level of security on devices and it describes a mechanism to associate users with their devices for remote credential provisioning.

Chapter 4 encompasses Publication II, Publication III and Publication IV. It presents the use of credentials stored on users' devices over low bandwidth channels such as NFC. In particular, it discusses an account-based ticketing architecture for NFC-enabled devices, and it presents various identity verification schemes that can be used in public transport ticketing.

Chapter 5 is based on Publication VI. It presents the use of TEEs to provide privacy guarantees in service infrastructure, such as cloud-based services. In particular, it describes how to design a privacy-preserving membership test using a TEE to ensure users' data privacy in a cloud-based malware checking service.

Finally, Chapter 6 presents different substantial aspects that are prevalent in current systems that use TEEs. These aspects are discussed based upon a synthesis of the previous chapters.

2. Background

This chapter presents relevant background information for the rest of the dissertation.

2.1 Smart cards

A smart card is typically a pocket-sized plastic card with an embedded micro-controller. It provides a tamper-resistant processing environment that isolates the runtime application states and data from an external environment [80]. It also facilitates secure storage for protecting persistent data stored on the card. Smart cards are widely used to host credentials such as public key based cryptographic tokens. Typically, smart cards are removable devices that communicate with a host device, such as a PC, through a card reader. A smart card can exhibit either a wired or a wireless interface, or both interfaces [80].

A suite of well-defined specifications is available for smartcard technology. The family of ISO/IEC 7816 standards describes the physical characteristics of smart cards, communication protocols, and interfaces for accessing/managing smartcard applications and data. Similarly, ISO/IEC 14443 and ISO/IEC 18092 describe the wireless interface and protocols for smart cards. Furthermore, Personal Computer/Smart Card (PC/SC) specifications [74] are designed to integrate smart cards on computing platforms such as PCs.

Smart cards were initially designed for a single application use, however, over time the smart card architecture has evolved into a complex operating system [80]. For example, MULTOS[1], is a smartcard OS that

[1]MULTOS `https://www.multos.com/`

is capable of securely hosting multiple applications [16, 36]. In multi-application smart cards, each application consists of a separate security domain that isolates the application resources from other applications on the card. The domain owner controls each security domain, and therefore, hosting an application in a card requires either the creation of a separate domain or the permission of a domain owner [36]. This architecture where the domain owner retains the right to the host application in a security domain is called the *issuer-centric* card ownership model.

The issuer-centric ownership model is inflexible for maintaining and updating services on the card once it has been issued. In other words, as a user joins new services and leaves existing services, it becomes impractical to add/remove applications on the card. *User-centric* card ownership models such as the *user centric owner model* (UCOM) [1], the *open provisioning* model[59] and the GlobalPlatform *consumer centric model* [34] have been proposed to overcome this issue. In the user-centric card ownership model, card issuers delegate ownership of the card to card holders. Cardholders have full permission to host applications of their choice on their cards [1].

Some smart cards such as Javacard and MULTOS also provide the application programming interfaces (APIs)[2] and tools[3] for application development and application management.

2.2 Trusted Execution Environment

A Trusted Execution Environment (TEE) [31] is a system security primitive that provides hardware-enforced isolation for executing security sensitive application logic. The TEE isolates its hardware and software resources from a general processing environment, also referred to as the rich execution environment (REE), where the majority of the platform software, including the primary operating system, runs. Figure 2.1 depicts the TEE system architecture defined by GlobalPlatform, a standards development organization that defines various TEE standards. An application logic that runs inside the TEE is referred to as a *trusted application* (TA), whereas the REE application, which initiates and interacts

[2]Javacard APIs: `https://docs.oracle.com/javacard/3.0.5/api/index.html`
[3]MULTOS SDK: `https://www.multos.com/developer_centre/tools_and_sdk/`

with the TA, is called a *client application* (CA).[4] The TEE can protect the confidentiality and integrity of a TA's runtime states and data from other applications running on the platform [33].

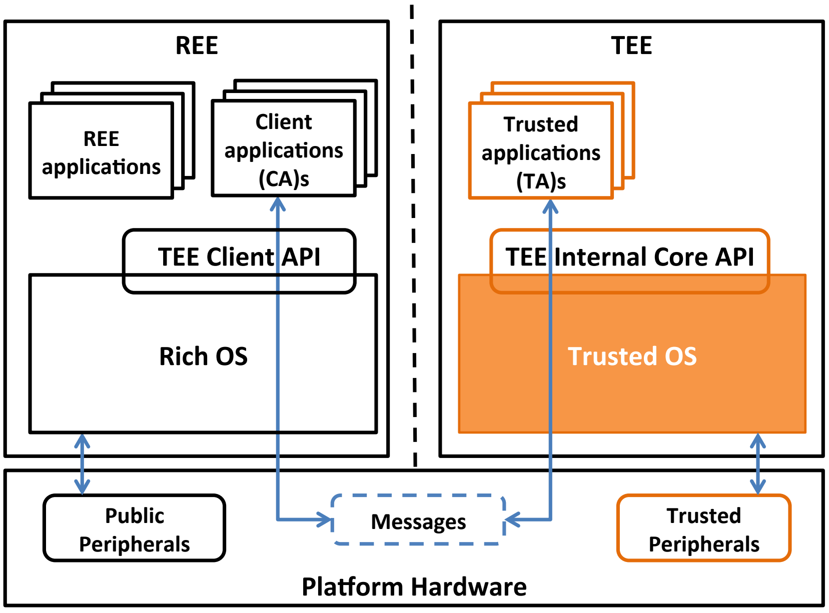

Figure 2.1. GlobalPlatform's TEE system Architecture

GlobalPlatform[5] has defined an extensive set of standards for TEE functionality, interfaces, and management. The *TEE System Architecture* [33] defines the hardware and software architecture that constitutes the TEE. The *TEE Internal Core API* [37] defines (C-programming) interfaces available within a TEE, such as memory management, cryptographic primitives and secure storage for the TA development. The *TEE Client API* [32] provides OS driver interfaces that activate a TA and allow a CA to establish a session with the TA for data exchange. Additionally, the TEE also allows CA and TA to share memory area to transfer large amount of data between them. However, GlobalPlatform recommends to limit this shared memory feature only for communication purposes as both CA and TA can

[4]CA is widely used as an abbreviation for *Certificate Authority*. However, this dissertation uses CA to refer the GlobalPlatform defined client application.
[5]www.globalplatform.org

modify the memory content asynchronously [33]. The shared memory is subject to the memory observation attacks by an adversary who controls the platform's REE. The *TEE Management Framework* [38] provides a framework for managing the TA lifecycle.

Modern CPU architectures include security primitives to augment a CPU with a TEE without the need for an additional hardware security component. The two most common TEE platforms available today (ARM TrustZone and Intel SGX) are described in the following subsections.

2.2.1 ARM TrustZone

ARM TrustZone [7] is the contemporary TEE architecture that is widely available on smartphone hardware platforms. TrustZone allows a single physical processor to split into two virtual processors, namely *secure world* and *normal world*. Each world consists of a runtime environment with its hardware and software resources. The secure world is the TEE and is used for processing security-sensitive application logic whereas the normal world is the REE that runs the main OS and normal applications.

Figure 2.2 shows an architectural overview of ARM TrustZone. By default, the boot sequence initializes the secure world operating environment first, after executing the initial bootloader from the SoC ROM [7]. The boot sequence then activates the normal world operating environment. ARM TrustZone also supports a *secure boot scheme* where each stage of the boot sequence verifies the signature associated with the next stage software before loading it into the secure world. The secure boot sequence establishes a chain of trust starting from the initial bootloader to prevent unauthorized or modified software from running into the secure world.

Once in the normal world, an entry to the secure world is controlled through a new processor mode called the *monitor mode*. The normal world calls a dedicated instruction called the *Secure Monitor Call* (SMC), which is handled by the monitor mode to switch to the secure world. The physical processor switches execution between the worlds in a time-sliced manner.

TrustZone allows configuration of hardware resources to be accessible in one of the two worlds [7]. It adds an extra control signal, called a

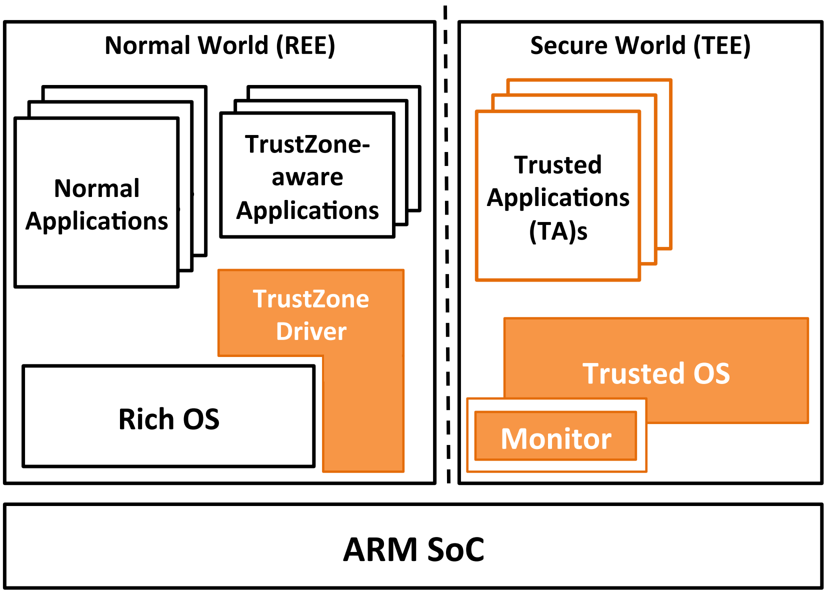

Figure 2.2. TEE with ARM TrustZone

Non-Secure (NS) bit, for each read/write channel on the main system bus, which determines the resources that can be accessed from the corresponding world. The NS bit value in a special register called the *Secure Configuration Register* indicates whether the processor executes in the secure or normal world at any given moment.

The TrustZone-aware memory management unit (MMU) isolates the main memory, such as Dynamic RAM (DRAM), between the two worlds by enabling a separate set of virtual-to-physical address translation tables for each world [99]. Such an MMU also allows each world to tag its entries in the translation look-aside buffer (TLB) with their identities i.e. using the NS bit [7]. The tag prevents flushing the TLB while switching between the worlds, resulting in faster context switching. TrustZone also isolates the internal memory (e.g. system-on-a-chip (SoC) RAM), available on the processor by partitioning it exclusively for the secure and the normal worlds' use. Unlike external memory, the internal memory within the secure world is both *secret* and *private*: TrustZone prevents the normal world from accessing secure world memory. Further, it prevents a normal world application from observing the memory access patterns of the secure world within its internal memory (i.e. the metadata regarding the memory locations and the order in which they are accessed).

TrustZone does not provide interfaces to communicate with a TA from REE applications [99]. However, several TEE providers run their secure OSs in the TrustZone TEE to facilitate interaction with a TA as well as TA development. The following paragraphs provide descriptions of two separate TrustZone-based TEE providers used in this dissertation.

On-board Credentials (ObC)

On-board Credentials (ObC) [59] is a TEE architecture developed by the Nokia Research Center, which was designed for two different hardware security extensions: M-Shield [89] from Texas Instruments and TrustZone [7] from ARM. ObC was available on Nokia Windows Phone 8 and Symbian phones. ObC provides runtime isolation for TAs using a small virtual machine called *ObC interpreter*. ObC is primarily designed for processors with a very small internal memory (i.e. SoC RAM). It allocates all the memory required for a TA from the available internal memory. Thus, the memory of TA can be regarded as both *secret* and *private*.

ObC supports a device specific key called the *ObC platform key* (OPK) protected in the TEE of the device. ObC derives application-specific *sealing/unsealing keys* from the OPK. TAs use sealing/unsealing keys to encrypt secret data before storing it persistently in the REE. This mechanism of encrypting data locally in the TEE and storing it in a persistent memory outside of the TEE is called *sealing*. ObC also stores TAs in an encrypted format in the REE. During execution, the *ObC scheduler* loads the encrypted TA into the TEE along with possible input parameters and sealed data, belonging to the TA, stored from its previous invocation.

ObC also includes a device-specific key pair (private/public) used for TA provisioning. The private key is always protected in the TEE. A trusted authority, such as the device manufacturer, certifies the public key during the manufacturing process. The device-specific key pair allows ObC to support an open provisioning where it allows developers to deploy their TAs on the TEE without requiring explicit permission from authorities.

TAs for ObC can be developed either in the BASIC programming language or in a custom bytecode assembly language. ObC also provides

APIs for provisioning new TAs into the TEE, executing a TA and interacting with a TA from REE applications. The ObC APIs also provide a set of standard cryptographic functions (e.g. encryption / decryption and signing / verification) that REE applications can use without requiring application-specific TAs [20].

Kinibi OS

Kinibi [97] is a micro-controller OS developed by Trustonic[6]. It is designed to run as the secure OS in ARM TrustZone. Kinibi isolates TAs by running each TA in a separate security domain. Each security domain consists of a domain-specific encryption key and an integrity protection key required for running the TA in the TEE.

In Kinibi, an external server called the Trusted Service Manager (TSM) assigns each TA a universally unique identifier (UUID). A TA provisioner must receive a device-specific license from the TSM to be able to provision a TA on the device. The license includes the domain-specific keys which are bound to the UUID of the TA. When the TA is provisioned, the license grants permission to run the TA on the device. Kinibi allocates a separate security domain for the TA and associates the domain-specific keys included in the license.

Kinibi limits the size of a TA to 1 megabyte (MB) which comprises its code, and its initialized and uninitialized memory in heap and stack. Kinibi allows configuring a TA to use either the internal memory of the processor or the main memory available on the device platform for memory allocation. When the internal memory is used, the TA is protected against an adversary observing the memory access patterns of the TA.

Kinibi also allows a CA to share up to 1 MB (in some hardware up to 6 MB) of main memory with a TA. The shared memory can be used to exchange information between the CA and TA. However, the shared memory is subject to memory observation attacks by an adversary who controls the REE. The interaction between a CA and a TA follows a request-response pattern where the CA can invoke any function supported by the TA.

Kinibi supports GlobalPlatform compliant TEE APIs for TA development and communication with REE applications. Additionally, Trustonic also provides proprietary TEE APIs.

[6]`https://www.trustonic.com/`

2.2.2 Intel Software Guard Extensions (SGX)

Intel's Software Guard Extensions (SGX) is an extension to the x86 CPU architecture that allows individual applications to execute selected code and data in protected containers called *enclaves*. Unlike TrustZone, SGX does not require separate TAs to run in a single platform-wide TEE. Instead, SGX allows applications to establish their own enclaves.

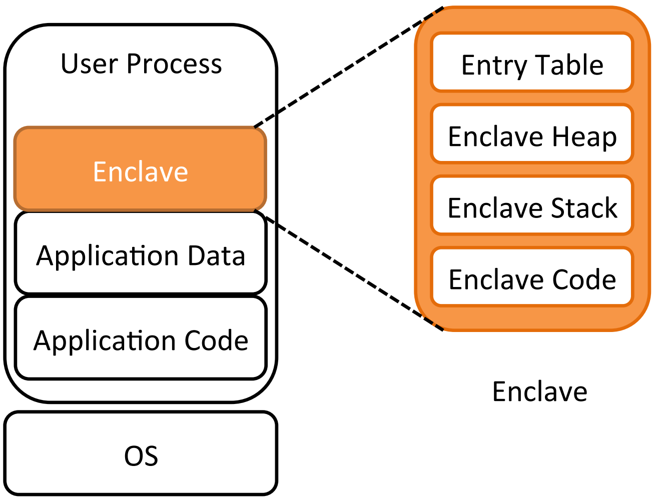

Figure 2.3. An enclave within the virtual address space of its host

As depicted in Figure 2.3, an enclave executes in the virtual address space of its host application, which enables the enclave to access the memory space of its host application. However, the SGX architecture prevents access to the memory of the enclave from the main OS, and all other applications including the host application and other enclaves. In addition, SGX automatically encrypts all enclave memory that leaves the CPU, such as while storing it into DRAM. SGX also maintains the measurements of the memory content as a log in a special register to ensure its integrity. Although the memory of an enclave in DRAM is secure (i.e. it provides confidentiality and integrity protection), it is not *private* as in ARM TrustZone. SGX cannot prevent an adversary from observing which memory pages the enclave reads or writes. This can lead to side-channel attacks [102, 64]. However, the adversary can only observe memory accesses at page-level granularity.

Intel provides an SDK that includes C/C++ APIs, libraries, documentation, and tools for application developers to create and debug enclaves.

2.3 Trusted Platform Module (TPM)

The Trusted Computing Group (TCG)[7] standardizes Trusted Platform Module (TPM) specifications [95, 96]. TPM specifications define security functionality for a module attached to a device platform that allows the device to prove its trustworthiness to a third party. A TPM can be implemented either as a discrete chip integrated on a device platform (TPM 1.2 or earlier), or as a part of the firmware of the platform, or as a TA [79] running in the platform TEE (TPM 2.0 onwards).

Unlike TEEs, TPMs do not permit application-specific logic to execute within a TPM. However, TPMs provide interfaces to allow the OS and other applications to access TPM functionality. TPMs can be used to build trustworthy systems. Since a TPM is bound to a particular device platform, it can be used to identify the device. Further, the TPM allows reporting its platform configuration to a third party, which allows to attest the device. TPM specifications are available in two versions namely TPM main specification (TPM 1.2) and the family of TPM 2.0 library specification.

2.3.1 TPM 1.2

A TPM contains several Platform Configuration Registers (PCRs) for storing aggregated measurements of events related to the security state of the platform. It also consists of non-volatile storage for protecting data from external entities. In TPM 1.2 [95], each TPM chip includes a permanent RSA key pair, called an *Endorsement Key* (EK), fabricated on the chip during manufacturing. It also includes a certificate for the EK, signed by a trusted Certificate Authority. Thus, the EK is used to identify and validate a TPM.

[7]Trusted Computing Group: `http://www.trustedcomputinggroup.org/`

A TPM creates a non-migratable key pair called the *Storage Root Key* (SRK) whenever a user takes an ownership of the TPM. The SRK is the root of a key hierarchy in the TPM. Due to privacy concerns, the EK is not used for attesting platform configurations. Instead, the TPM generates a second key pair, called an *Attestation Identity Key* (AIK) for attestation purposes. A privacy certificate authority (PCA) authenticates the public key of AIK with respect to EK and issues an anonymized certificate for an AIK. The TPM digitally signs its PCR values with the AIK and sends the signature to the requester for attestation. TPM 1.2 also supports Direct Anonymous Attestation (DAA) [12] to produce anonymous signatures. However, DAA is marked as optional function in TPM 1.2.

TPMs also allows *binding* encrypted data locally to a TPM using the SRK which can only be decrypted by the TPM. Further, *sealing* ties the encrypted data to a specific platform measurements (PCR state).

2.3.2 TPM 2.0

Unlike TPM 1.2, the TPM 2.0 [96] generates the EK using a TPM-specific Endorsement Primary Seed (EPS). Instead of using AIK, TPM 2.0 uses DAA to provide anonymity. Further, the specification does not tie the TPM to a particular cryptographic algorithm, e.g. it supports elliptic curve cryptography (ECC) in addition to RSA. Additionally, it features algorithm agility, which allows manufacturers to implement any cryptographic algorithm from the list approved by the TCG. TPM 2.0 also allows replacing the ESP with a new random value, in which case it invalidate all certificates associated with the EK generated from the previous EPS value. Similarly, TPM 2.0 generates the SRK from a Storage Primary Seed (SPS). Additionally, TPM 2.0 includes a Platform Primary Seed (PPS), which is used to generate a key hierarchy exclusively designed for platform firmware use. TPM 2.0 further introduces *Enhanced Authorization* (EA) policies that provide access control on objects protected by the TPM through user-defined policies.

2.3.3 TPM on mobile device platforms

TPMs have been available on PC platforms for some time. Recently, they have also become available on other device platforms, such as mobile phones. Microsoft requires all new device platforms designed for its Windows 10 OS to include TPM 2.0 module.[8] Microsoft Research exposes TPM 2.0 functionality to application developers through the TPM Software Stack.[9] Also, TPM 2.0 module can be implemented as a TA running in the platform TEE of mobile devices [79].

2.4 Remote Attestation

Remote attestation is a mechanism for a device to prove its hardware and software configuration to a remote verifier. The device uses trusted hardware, such as a TPM, to sign the attestation information. The TPM signature allows remote verifiers to trust the attestation information because the TPM key used to produce the signature is certified by a trusted certificate authority [95].

TPM-based attestation is designed to provide attestation of the entire software configuration of the device, e.g. to all software loaded since the device was booted. However, attestation of an individual application with a TPM is not straightforward. Furthermore, the remote verifier requires information about all possible software configurations for each device. However, current common practice is for software and applications to be continuously added, removed or updated. Thus, it is impractical for a verifier to maintain a list of all possible software configurations. Also, providing the entire software configuration to a remote verifier can become a privacy concern since the remote verifier can infer personal traits of the device user based on the software installed on the device [58].

Some TEEs, such as Intel SGX, allow individual applications to perform remote attestation. In contrast, TrustZone does not facilitate remote attestation by default. Some TrustZone-based TEE providers have implemented mechanisms for remote attestation. The following subsections describe remote attestation in TrustZone-based TEEs ObC and Kinibi, and Intel SGX.

[8] https://msdn.microsoft.com/library/windows/hardware/dn915086(v=vs.85).aspx
[9] https://github.com/Microsoft/TSS.MSR

2.4.1 Remote attestation in ObC

Remote attestation in ObC is based on a certificate chain, the root of which is the certificate issued by the certificate authority of the original equipment manufacturer (OEM). Each device contains a device-specific signature key pair. The device manufacturer issues a device certificate for the public part of the signature key during the manufacturing process. The manufacturer's signature key, used to produce device certificates, is in turn certified by the certificate authority of the OEM. The device certificate includes information such as the identity of the device and its TEE.

The private part of the signature key is stored securely in the TEE and maintained by a *subCA* unit of ObC. The subCA is responsible for generating a certificate for a TA-specific key pairs created using ObC APIs. A TA can request ObC to generate a key pair, to certify the public key and use the keys. However, the TA does not have access to the private key object itself. ObC also prevents TAs from directly interacting with the subCA to prevent certification of keys that are either generated outside ObC or generated using a mechanism other than ObC APIs.

During key pair certification, the subCA includes private key metadata (e.g. access control attributes, key usage attributes) into a *subject key attestation evidence* (SKAE) structure [94]. The SKAE also includes the identifier of the TA that requested the key pair, and the hash of the public part of the key pair. The subCA adds the digest of the SKAE to the key pair certificate as an extended certificate attribute.

The remote verifier validates the certificate chain starting with the root certificate issued by the OEM to the key pair certificate of the TA. The validation of SKAE attributes affirms the correctness of the TA, assures that the key pair was generated in the TEE with ObC APIs, and assures that the TA has usage access to the keys.

2.4.2 Remote attestation in Kinibi

Kinibi uses an attestation server [19], maintained by the TEE provider, during remote attestation. The attestation server shares a unique symmetric key, called the *device root key*, with each device. Kinibi provides an attestation service interface for TAs to request data attestation.

To attest a key pair generated in the TEE, a TA provides the digest of the public key to the attestation service interface. Kinibi uses the device root key to sign the public key digest along with the UUID of the TA, the identity of the TEE itself and the device identity. The signed data is encrypted using the public key of the attestation server and sent to the attestation server, which validates the signature and provides attestation.

The attestation asserts the existence of the TEE in the device, assures that the TA with mentioned UUID is running in the TEE and that the TA has access to the key pair. The Kinibi attestation does not limit TAs to attest only key pairs but also allows the attestation of any data. (Remote attestation in Kinibi is briefly explained in Section VI.B of Publication V.)

2.4.3 Remote Attestation in SGX

SGX allows an enclave to attest its identity to another enclave running on the same platform through *local attestation*. SGX allows the enclave to include messages in the attestation request. SGX produces an *attestation REPORT* [3], which includes a Message Authentication Code (MAC) that binds the enclave's messages with the enclave's identity and its measurements (i.e. code and data of the enclave). SGX computes the MAC using a symmetric key, which it shares only with the target enclave that requires verification of the attestation REPORT [14].

Additionally, SGX also enables remote attestation of an enclave. SGX involves a dedicated enclave called the *Quoting Enclave* for remote attestation. During remote attestation, an enclave sends attestation REPORT to the Quoting Enclave. The Quoting Enclave first attests the REPORT using the local attestation. Then, it replaces the MAC with an *attestation signature* generated with a device-specific attestation key. To provide anonymity during remote attestation, SGX uses a group signature scheme called Enhanced Privacy ID (EPID) [13], which is an extension of the DAA scheme. A device platform receives an EPID Member Private Key as the attestation key from the Intel's key provisioning service using a dedicated Provisioning Enclave. SGX only allows the Quoting Enclave to access the EPID Member Private Key for generating attestation signatures [14].

2.5 Electronic Identity

Governments around the world have been issuing digital credentials or Electronic Identity (eID) to their citizens for over a decade. Typically, a government-issued eID is in the form of a smart card that holds the credentials of a citizen inside a secure chip and other details about the citizen, such as a picture, name, date of birth, etc., printed on the surface of the card. Thus it can be used for both digital and physical identification of its holder.

The European Union Electronic Signature Directive (1999/93/EC) [17] uses the term *electronic signature products* (ESPs) to describe the trusted endpoint that carries out an eID transaction. An ESP is a logically isolated security token that produces digital signatures to prove the identity of its user. The user controls the ESP, and it often resides with him or in his device. The function of the ESP has been integrated on smart cards as well as on SIM (subscriber identity module) cards.

Most of the eIDs contain two types of signatures using separate keys: a qualified signature key for document signing and an authentication key for proving the identity of the holder. The authentication key can also be used to decrypt data that has been encrypted with the corresponding public key.

2.6 Near Field Communication

Near Field Communication (NFC) is a short-range radio-frequency communication technology operating at a frequency of 13.56 MHz. It supports data rates of 106 kilobits/s, 212 kilobits/s, or 424 kilobits/s. NFC comprises a set of wireless communication standards that define protocols and mechanisms to establish radio communication between two devices within proximity, typically up to a distance of 10 centimeters.

The family of ISO/IEC [45], ISO/IEC 15693 [46] and Japanese JIS 6319-4 [53] (FeliCa[10]) define the physical characteristics, radio-frequency signal interface, initialization schemes and transmission protocol for proximity integrated circuit cards (PICC) such as contactless cards. PICCs are also called *passive devices*, which do not include a power supply of their own.

[10]`http://www.sony.net/Products/felica/`

ISO/IEC 18092 [49] defines the physical characteristics of proximity coupling devices (PCDs) such as NFC readers. PCDs are also called *active devices* that constitute power sources, such as batteries, to generate a radio field for communication. ISO/IEC 21481 [48] specifies the mechanism for an active device to detect and select a passive device within its proximity for communication.

NFC Forum[11] is a consortium of different organizations that standardizes NFC technology and encourages interoperability between NFC devices. It has published various NFC specifications[12] including NFC data exchange formats (NDEF) and NFC tag formats for NFC applications.

During communication, an active device activates a passive device and initiates communication with it. A passive device, within the proximity range of the active device, harvests power from the electromagnetic field generated by the active device. Passive devices always operate and respond to the requests sent from an active device. On the other hand, an active device can operate in one of the following three different modes:

1. **Reader/Writer mode:** In this mode, an active device interacts with a passive device to read or modify data stored on the passive device. The family of ISO/IEC 14443 standards defines the protocols for establishing NFC channels and exchanging data between the active and passive devices. For example, a contactless reader (operating in the reader/writer mode) at a transport endpoint can read the ticket information stored in a travel card.

2. **NFC peer-to-peer mode:** In this mode, two active NFC-enabled devices establish a bi-directional communication channel between them for data exchange. The logical link control protocol (LLCP)[13] defines the data link layer protocol necessary for NFC peer-to-peer communication. For example, two NFC-enabled mobile phones in proximity exchange contact information over an NFC interface using the peer-to-peer mode.

3. **Card emulation mode:** In this mode, an active NFC device em-

[11] http://nfc-forum.org

[12] http://nfc-forum.org/our-work/specifications-and-application-documents/specifications/nfc-forum-technical-specifications/

[13] http://nfc-forum.org/our-work/specifications-and-application-documents/specifications/nfc-forum-technical-specifications/

ulates those of a passive device to communicate with other active NFC devices. The emulating device exhibits interfaces and communication parameters identical to those of a passive device making it virtually indistinguishable when it is read from or written by an external active device. For example, a ticketing application on the NFC-enabled mobile phone emulates a travel card to interact with contactless readers at transport endpoints.

2.7 Public transport ticketing

All over the world, for years transport ticketing has been implemented using contactless cards also known as *travel cards*. Travel cards primarily use proximity card technology such as ISO/IEC 14443 and FeliCa systems.

A travel card consists of a *ticketing application* that stores ticket information such as the remaining balance, validity period and card holder's details. *Checkpoint terminals* (usually contactless card readers) at the transport system end-points (e.g. at the metro stations) verify ticket information on cards and deduct ticket fares before allowing card holders (*travelers*) to travel.

Ticketing applications are secured using protocols such as Mifare[14]that allow proximity cards such as ISO/IEC 14443 to be used as secure tokens or travel cards. Mifare-powered ticketing systems have proven to be very usable in practice, despite some vulnerabilities [29, 28, 15].

[14]http://mifare.net

3. Securely provisioning and using credentials on devices

This chapter describes how to securely provision and store credentials on a user's device. In particular, it focuses on provisioning and using smartcard-based credentials on users' devices such as smartphones.

Credentials stored on smart cards: Smart cards are widely used to store credentials securely. Typically, *credential issuers* provision credentials to smart cards before distributing the cards to their users. For example, some governments issue electronic identity (eID) credentials to their citizens in the form of smart cards. eID credentials can be used as electronic signature products (ESPs) to prove the identities of card holders (Section 2.5). In such credential distribution models, the binding between a smartcard-based credential and the identity of a user occurs within the premises of the credential issuer. Users can use credentials stored on smart cards via a card reader connected to their personal computers (PCs). For example, a user can authenticate herself to access an e-health service from a web browser on a PC using her government-issued eID card connected to the PC through a card reader.

One of the major disadvantages of smart cards is the lack of a trusted path [104] for the user; the lack of user interface (UI) on the card surface constrains users to type in their PINs on the UI displayed on host PCs when using the credentials. Thus an adversary on the PC can spoof the PIN-entry UI, or listen to the communication channel between the reader and the PC to steal the PINs, and make unauthorized credential accesses whenever the card is connected. Furthermore, smart cards are inflexible in the sense that the cards often only serve a single purpose and are unusable once the service expires. Therefore, a user subscribing to multiple services from different service providers ends up with many cards that need to be carried everywhere. The user may also need to carry a card reader to be able to use services from conventional host devices that are not equipped with built-in card readers.

Credentials stored on users' devices: On the other hand, users' devices such as smartphones provide a direct interface for the users. As a separate unit, the UIs on the users' devices can act as a trusted interface to protect against unauthorized use of the credentials hosted on these devices. For example, the device can display a PIN-entry UI before using the credentials. Additionally, the device platforms have sufficient storage capacity and computing power to host many credentials on a single device. They are also more convenient than physical smart cards. They are always within the vicinity of users, and various standards are available to establish communication channels (e.g. Bluetooth, USB, etc.) between a user's device and a PC. Therefore users do not need to carry an additional card reader. Credentials can also be provisioned remotely, reducing the cost of deployment.

3.1 Hosting credentials on users' devices

Hosting smartcard-based credentials on user's devices is not a novel concept. Already almost two decade ago, Balfanz et al. [8] proposed a software smart card architecture where the smart card functionality was implemented on a Personal Digital Assistant (PDA) rather than as a separate hardware module. Although the user's device protects against unauthorized use of credentials from a host PC, the software smart card architecture alone is insufficient to prevent unauthorized use of credentials from applications running on the same user's device.

Over the last decade, the state of technology has advanced significantly such that modern smartphones have replaced the dedicated, slow, unconnected PDAs of the late 90's. Smartphones are in essence very close to personal computers when it comes to openness and feature richness. However, the security of the smartphone OS have been routinely defeated using techniques such as *jailbreaking* on iOS and *rooting* on Android OS [99]. Also, mobile malware is on rise [27] Similarly, mobile malware is rising — an anti-malware vendor, G Data, in 2015 reported approximately 2.3 million new Android malware samples, which was a 50 % increase in mobile malware from the year before [27]. Although mobile malware samples are increasing, Truong et al. [93] indicated that the actual malware infection rates are very low. This may be due to the platform security deployed on smartphones, which are designed to be more secure than PCs [60, 99]. Nevertheless, credentials hosted on a user's device must be protected against malicious applications including the malicious OS.

Unlike smart cards, credential issuers do not control users' devices. Further, it cannot be expected that credentials are provisioned to a user's device during device manufacture. Therefore, credential issuers require trust guarantees that the user's device fulfills the security requirements for hosting credentials. They also need assurance that the device to which they issue credentials belongs to the legitimate user. However, the software smart card architecture does not provide remote attestation to assure the level of security available on the user's device. Furthermore, it does not describe how to associate devices with their users, i.e. user-to-device binding.

At present, trusted execution environments (TEEs) are available on modern device platforms, such as smartphones [21]. As a TEE provides a secure, isolated execution environment for processing security-sensitive application logic, it can act as a trust anchor to fulfill the security requirements for hosting credentials on users' devices [21, 99]. Therefore, a hardware-supported virtual smart card (VSC) architecture can be designed in such a way that the smart card functionality is implemented as a trusted application (TA) running inside the TEE.

3.2 Research Questions

Based on Publication I and Publication V, this chapter investigates the following research questions for designing a VSC architecture that allows remote provisioning of credentials and hosting different types credentials on a single user's device.

RQ 1. Can modern devices, such as smartphones, be a better platform for hosting credentials compared to dedicated hardware tokens or smart cards, in terms of hosting *multiple* credentials on a single user's device, and providing a user interface for authorizing credential use?

RQ 2. During remote credential provisioning, how to attest the level of security on the users' devices to the remote credential issuers?

RQ 3. How to associate users' identities with the devices to which the credentials are issued?

3.3 Methodology

The methodology used to answer the above research questions involved is as follows:

a) designing a hardware-supported VSC architecture and a remote credential provisioning architecture
b) implementing the architectures on users' devices with TEEs
c) developing a reference implementation of a real smartcard-based credential (Finnish electronic identity)
d) evaluating these systems in terms of security and usability.

Before describing the system architecture, it is important to understand the adversary model and define the requirements to protect the system from the adversaries.

3.3.1 Adversary model

There are two types of adversaries: (1) a *local adversary* who has physical access to and full control of the REE on the user's device and (2) a *remote adversary* who does not have physical access to the device but can execute arbitrary code on the REE of the device, e.g. through malware. The local adversary may also attempt sophisticated side-channel attacks such as cache attacks [63] and rowhammer [98] against the device's TEE. However the cost of performing such attacks is comparatively more expensive than the value of the data it can extract. Therefore, it is assumed that neither of these types of adversaries can undermine the security guarantees of the TEE. These adversaries have the following objectives:

- **Credential manipulation:** The device owner is a local adversary who intends to manipulate the credential information, e.g. rollback expired credentials, to gain additional benefits.

- **Unauthorized credential use:** The local adversary also intends to make unauthorized service accesses with the credentials on the device. Similarly, the remote adversary could also make unauthorized service access requests using the device credentials via malware.
- **Cloning credentials:** The local adversary may also intend to clone credentials from the user's device onto another device. Similarly, malware on the device could copy and deliver credential information to the remote adversary.
- **User Impersonation:** The remote adversary may impersonate a legitimate user to provision credentials onto the adversary-controlled device.

It is assumed that the device always establishes secure communication channels with credential issuers and service providers using correctly implemented cryptographic protocols, e.g. using standard TLS connections. The communication channel between a device and a host PC is assumed to be secured using an end-to-end physical connection such as a USB cable. Also, it is assumed that a local adversary other than the user herself cannot observe or interfere with the user while she is operating her device during credential provisioning.

3.3.2 Requirements

The following requirements apply to the design of the above mentioned architectures:

R3.1. **Integrity:** The VSC architecture should protect against unauthorized modification of credential information and should guarantee the correctness of credential operations.

R3.2. **Confidentiality:** The VSC architecture should only allow the correct TA to access credential secrets (e.g. private keys, PIN information). The main OS running on the device, and all other applications including other TAs should not be able to learn credential secrets.

R3.3. **Access control:** The VSC architecture should enforce access control to prevent unauthorized use of the credentials. For example, verifying the PIN entered by users before performing any operations on credentials.

R3.4. Device attestation: The remote credential provisioning architecture should include a mechanism to attest the device's security level to credential issuers.

R3.5. User-to-device Binding: The remote credential provisioning architecture should assure a credential issuer that the user's device to which it issues credentials belongs to the correct user.

R3.6. Flexibility: The VSC architecture should allow hosting different types of credentials, including credentials with proprietary protocols (e.g. Public transport ticketing, access control to a building, etc.) on a single device.

R3.7. Compatibility: The applications on the PC platforms should be able to access VSC credentials without any modification of the existing smartcard middleware. Similarly, other applications, such as web browsers, email clients, etc., on the user's device should be able to access credentials using standard interfaces.

These requirements do not include *availability*, as TEEs do not guarantee availability. The adversary on the device prevents users from accessing VSC services. It can even modify or delete information stored on the REE of the devices. Preventing such denial of service attacks are not considered in this thesis.

3.4 Hardware-supported VSC with remote provisioning

This section gives an overview of the proposed hardware-supported VSC architecture, explains the procedure for remote provisioning of credentials, and describes the implmentation of these systems. Full details are presented in Publication I and Publication V.

3.4.1 Hardware-supported VSC Architecture

Section 4 of Publication I presents a hardware-supported VSC architecture that uses a TEE on a user's device to host different types of credentials. The overall architecture is depicted in Figure 3.1. The VSC architecture includes a trusted application called VSC-TA, running in the TEE. The VSC-TA includes security-critical functionality of a smart card, e.g. cryptographic operations and PIN verification. The architecture also

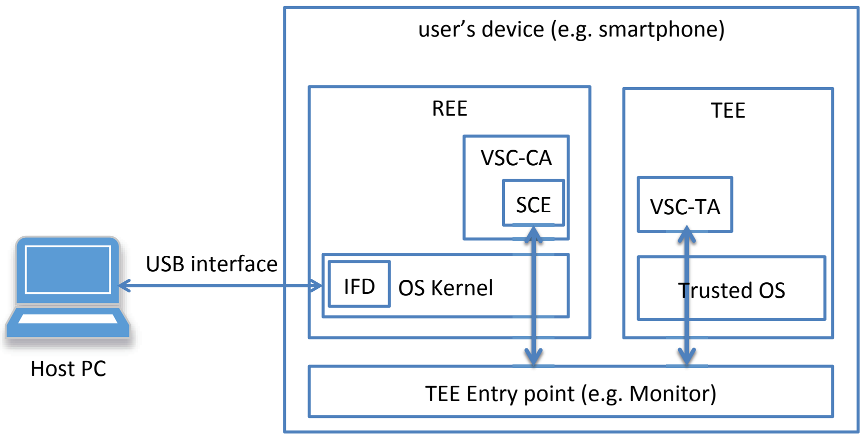

Figure 3.1. A hardware-supported virtual smart card architecture

includes an REE application called VSC-CA.

A VSC-CA consists of a smart card engine (SCE) module which is responsible for maintaining multiple VSC instances (a real world equivalent of a traditional smart card). The VSC-CA provides a user interface for initializing, provisioning, PIN entry, and selecting VSC instances. Since the secure storage capacity within a real TEE is limited, a VSC-TA can store credentials outside the TEE in encrypted form using the TEE-specific sealing mechanism. Sealing allows the VSC architecture to prevent adversaries from cloning VSC credentials as the sealing key is only accessible to the VSC-TA on the particular device.

During credential use, the VSC-CA transfers encrypted credential secrets of the selected smart card to the VSC-TA. To prevent against unauthorized credential use, the VSC-TA requires users to endorse the operations that use the credentials via a PIN. If PIN verification fails, the VSC-TA decrements the maximum PIN-retry counter value and allows users to retype the PIN until the retry counter reaches zero. After which it blocks all operations on the credential. PIN retry policies, as well as the PIN recovery mechanisms, are defined by the credential issuer during credential provisioning.

The architecture also consists of an Interface device (IFD) module that emulates a USB smart card reader interface. It allows external devices,

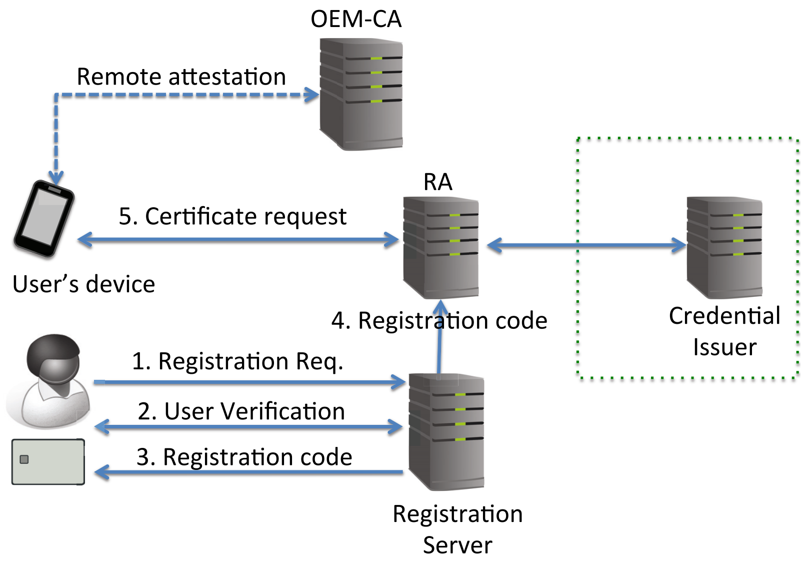

Figure 3.2. An architecture for remote provisioning of credentials.

such as PCs, to communicate with the selected VSC instance using standard protocols such as ISO/IEC 7816-4 [50] over USB channels. The IFD module can be extended to support other communication interfaces that allow PCs to communicate with the selected VSC instance, e.g. using an NFC interfaces.

3.4.2 Remote provisioning of credentials

Before provisioning credentials, users should download and install a VSC application onto their devices, e.g. from an application store. The VSC application consists of both the VSC-CA and the VSC-TA. The VSC-TA is loaded into the TEE whenever the VSC-CA application executes.

The remote credential provisioning architecture uses remote attestation mechanisms offered by the TEE providers. When the application runs for the first time, the VSC-TA creates an asymmetric key pair and requests the TEE to attest the public key. The TEE ensures that the private key is only accessible to the VSC-TA. The attestation assures the credential issuers that the key was generated in the TEE using a TEE-defined mechanism and that the key is only accessible to the TA that has requested the credential provisioning.

The provisioning architecture uses an existing strong digital identity of the user, such as a government-issued eID card or a mobile certificate, to associate the device with the user's identity. Figure 3.2 depicts an architectural overview of the remote credential provisioning. The user first authenticates herself to a registration server using her strong digital identity (Step 1 and 2). For example, this might include using a browser in a PC that can access her government-issued eID card. After authenticating the user, the registration server returns an ephemeral, one-time registration code to the user (Step 3), e.g. the server displays a registration code as a QR-code on the user's web browser. The registration server also submits a copy of the registration code along with the user's identity to the registration authority (RA) (Step 4).

In the next step, the user makes the registration code accessible to the VSC-CA application, such as by scanning the QR-code with the device camera. The VSC-CA then establishes a secure connection with the RA and begins the credential provisioning procedure (Step 5). The VSC-CA submits the attestation information (generated by the VSC-TA) for the public key to the RA. As explained in Section 2.4, the attestation mechanism depends on the device TEE and may involve the certificate authority of the TEE manufacturer, i.e. OEM-CA. The VSC-CA also submits a certificate signing request (CSR) for the public key along with the registration code to the RA. The RA first verifies the attestation information (e.g. using the public key of the OEM-CA certified by a trusted authority). Then, the RA performs a challenge-response protocol with the VSC-TA via the VSC-CA to ascertain the possession of the private key.

The provisioning architecture relies on users to transfer the registration code embedded in the QR-code to RA via the VSC-CA. If the adversary influences the VSC-CA to provide an incorrect registration code to the RA the provisioning process fails. Nevertheless, the malware on the device can forward the registration code to a remote adversary. If the remote adversary manages to complete registration before the user, the user notices provisioning process failure and can take correct measures (e.g. authenticate herself to the RA with her strong digital identity in order to deactivate the recently provisioned credential.

Also, to mitigate the threat of a requester impersonation, the RA ignores the user's identity normally embedded in the CSR template and replaces it with the user's strong identity, known to it during initial user authenticating, before forwarding the CSR to the certificate authority of the credential issuer. The certificate authority issues a certificate for the key, which can then be used as an ESP credential.

Section VII of Publication V provides further details of the remote credential provisioning architecture. Section 4.2 of Publication I further describes a smart card template provisioning protocol that allows the VSC-CA to download pre-initialized VSC. It also allows the VSC-CA to download plugins to support smart card templates with proprietary protocols, e.g. public transport ticketing protocols.

3.4.3 Implementation

The VSC architecture was implemented on two different device platforms: a Nokia X6 running Symbian OS[1] and a Nokia N900 running Linux OS (Maemo 5[2]). On-board Credentials (ObC) [59] provided the TEE functionality on both devices. The IFD module was implemented as a USB gadget driver[3] that ran at the OS kernel level. Section 5 of Publication I presents the implementation details of the VSC architecture. It also presents a reference implementation of a VSC instance based on the Finnish electronic identity (FINeID) card specifications [100].

The remote provisioning of credentials was also implemented on two different devices platforms with separate TEE providers: a Nokia Lumia 1520 running Microsoft Windows Phone 8 with ObC as the TEE, and a Samsung Galaxy Tab 3 running Android 4.2 with Kinibi as the TEE. Section VIII of Publication V presents the implementation details of the remote provisioning of credentials. It also provides a reference implementation for provisioning a credential based on FINeID.

Table 1 in Publication I and Table I in Publication V provide comparison of the performance measurements for the cryptographic operations used in VSC application implemented in TEEs (VSC-TA) and physical smart cards. The results show that operations on VSC credentials take less time

[1]Symbian Operation System `http://www.symbianos.org/`

[2]Maemo `http://maemo.org/`

[3]Linux-USB Gadget API Framework `http://www.linux-usb.org/gadget/`

in a device's TEE than similar operations on physical smart cards. Section 6 of Publication I provides security analysis of the VSC architecture.

3.5 Discussion

Integrity and Confidentiality of credential secrets: The VSC architecture utilizes a TEE to protect the integrity and confidentiality of the credential secrets stored on the user's device. The VSC-TA only stores encrypted credential secrets on the REE using a secret key that is accessible to it within the TEE (**requirements R3.1** and **R3.2**).
Correctness of credential operations: The TEE verifies the hash of the TA code against the digital signature (signed by the TA provider) before executing the TA. Unlike Kinibi, ObC allows a modified TA to execute. However, it treats the modified TA as a separate TA and prevents it from accessing the resources of the unmodified TA. Further, by definition, the TEE prohibits any REE application or the OS from subverting the behavior of a TA. Therefore, the TEE guarantees that the VSC-TA always operates on credentials correctly (**requirement R3.1**).
Access control on credential use: To prevent unauthorized credential use, the VSC-TA requires endorsement from users via a PIN before performing any operations using the credential (**requirement R3.3**). Unlike with smart cards, users are not forced to type in their PINs on a host PC but enter PINs directly on the PIN-entry UI displayed on their devices. Since the TEEs used for implementing the VSC-TA do not have a trusted UI, the VSC architecture displays the PIN-entry UI from the REE application, i.e. VSC-CA. Therefore, malware on the user's device can spoof the PIN-entry UI to capture the PIN. TEEs with trusted UI capabilities that prevent malware spoofing the PIN-entry UI are becoming available on new devices [18]. Trusted UI is further discussed in Section 6.3
Remote attestation: The remote attestation feature of TEEs provide assurance to the credential issuer regarding the security level of the device that requested the credential provisioning (**requirement R3.4**).
User-to-device Binding: The remote credential provisioning architecture (Section 3.4.2) utilizes the user's existing strong digital identity to bind the identity of the user with her device that has requested the cre-

dential provisioning (**requirement R3.5**). This mechanism can be easily integrated into the provisioning infrastructure of physical smartcard-based credential issuers to allow remote credential provisioning on the devices of their existing users.

Flexibility: The VSC architecture is designed in such a way that it allows the hosting of different types of credentials on a single device. The VSC-CA provides UIs to manage credentials and allows the selection of a specific credential to use (**requirement R3.6**).

Compatibility: The IFD module emulates USB interfaces of a physical USB smart card reader. Thus, a PC connected to the device over a USB channel can use credentials on the device in a similar manner as it uses a physical smartcard-based credential through a USB card reader without requiring any additional software or modification of the existing smartcard middleware (**requirement R3.7**). Publication I and Publication VI do not address mechanisms for using the VSC credentials from other REE applications on a device. However, other REE applications can directly communicate with the VSC-TA to use the credentials, provided that they are aware of the interfaces to communicate with the VSC-TA.

Alternatively, the VSC-CA can be designed as a service that provides standard cryptographic APIs through which other applications can use the credentials. For example, in Android, the VSC-CA can implement the Java Cryptography Extension (JCA)[4] to deliver a *Keystore provider* with similar interfaces as the Android Keystore APIs[5]. Other applications can use the Keystore provider to interact with the VSC credentials instead of the using credentials from the Android Keystore. Both methods require other applications to be aware of the available interfaces to use the credentials. However, it is usually impractical to modify existing applications to use VSC credentials.

On the other hand, *web applications* can be designed in such a way that they are aware of a service on the device that allows them to use VSC credentials. For example, the Finnish Population Register Center[6] has developed a Signature Creation Service (SCS) specification [61] that al-

[4]JCA Reference Guide: https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html

[5]Android Keystore System: https://developer.android.com/training/articles/keystore.html

[6]Finnish Population Register Center: http://vrk.fi/en/frontpage

lows HTML5 applications on a standard web browser to communicate with the SCS signer application[7] using the interfaces as defined in the Cross-Origin Resource Sharing specification[8]. The SCS signer application runs as a service to generate digital signatures using smart cards issued by the Population Register Centre connected to the device via a USB card reader. Similarly, the VSC-CA has been designed according to the SCS specification to allow web applications to use credentials without modification or the additional extension of a standard web browser [71].

Limitation: The devices used to implement the VSC architecture lack secure storage to protect against replay attacks. The TEE is unable to distinguish the most recent version of the encrypted TA data stored on the user space from earlier versions. Therefore, an adversary can simply replace the TA data with an earlier version to mount a replay attack. For example, the VSC application may only allow up to three PIN retries after which it locks down to protect against unauthorized accesses. However the adversary can simply replace the TA data with an earlier version to continue PIN retries. Recent TEEs, such as OP-TEE[9], make use of the replay protected memory block (RPMB)[10] partition of an embedded multimedia card (eMMC) to provide secure storage. The replay protected secure storage allows the VSC architecture to prevent against credential manupulation.

3.6 Conclusions

Modern devices, such as smartphones, are definitely better platforms for hosting credentials compared to smart cards (**RQ 1**) for the following reasons: (a) The TEE provides the necessary security guarantees for hosting credentials comparable to the smart cards. (b) The SCE module of the VSC-CA is designed to handle multiple VSC credentials and provides UI for selecting a credential for use. (c) Unlike smart cards, the users enter

[7]SCS app: https://play.google.com/store/apps/details?id=fi.fineid.security.scs

[8]Cross-Origin Resource Sharing https://www.w3.org/TR/cors/

[9]Open Portable Trusted Execution Environment (OP-TEE) https://www.op-tee.org/

[10]https://github.com/OP-TEE/optee_os/blob/master/documentation/secure_storage_rpmb.md

their PINs directly on their own device UI to protect against unauthorized credential use from host PCs. Indeed, with the availability of trusted UI features in modern TEEs, the credentials can be further protected from unauthorized use by other applications on the user's device. (d) Hardware-supported VSCs enable faster credential operations compared to physical smart cards. The remote attestation mechanisms offered by TEEs assure the credential provider of the level of security available on devices (**RQ 2**) and the mechanism of using existing physical credentials fulfills the user-to-device binding requirement (**RQ 3**).

The feature-rich environment of users' devices (specifically the user interface and communication capability) extends the usage of credentials beyond a physical smartcard infrastructure to meet the demands of today's distributed systems, i.e. credentials can be provisioned, updated, and maintained remotely upon request. Thus, credentials can be deployed and used more efficiently than the physical smart cards at relatively low cost.

4. Securely using credentials over low-bandwidth channels

Chapter 3 presented a hardware-supported virtual smart card architecture for securely hosting different types of credentials on a user's device and using them from a host PC over a USB channel. This chapter describes how to use such credentials stored on users' devices directly over low bandwidth channels such as NFC.

In recent years, mobile phones equipped with NFC technology have become widely available, which has stimulated the relocation of applications traditionally hosted on contactless cards to users' devices, including, for example, small value payments, public transport ticketing and access controls for buildings.

A decade ago, public transport ticketing on NFC-enabled phones was already deployed in Japan [30]. Public transport systems around the world use contactless cards, also called travel cards, as transport tickets. A travel card consists of a *ticket application* that stores the ticket information such as the ticket balance, validity period, and card holder's details.

Ticket applications are usually proprietary and are incompatible among different transport systems. Attempts such as ITSO specification [51] have been designed to make transport ticketing more inter-operable between different transport systems. However, the majority of the public transport systems have not yet adopted the ITSO specification. The Open Ticketing Institute (OTI)[1] attempts to overcome the ticket interoperability barrier through an *account-based ticketing* approach.

[1]Open Ticketing Institute `http://www.openticketing.eu/`

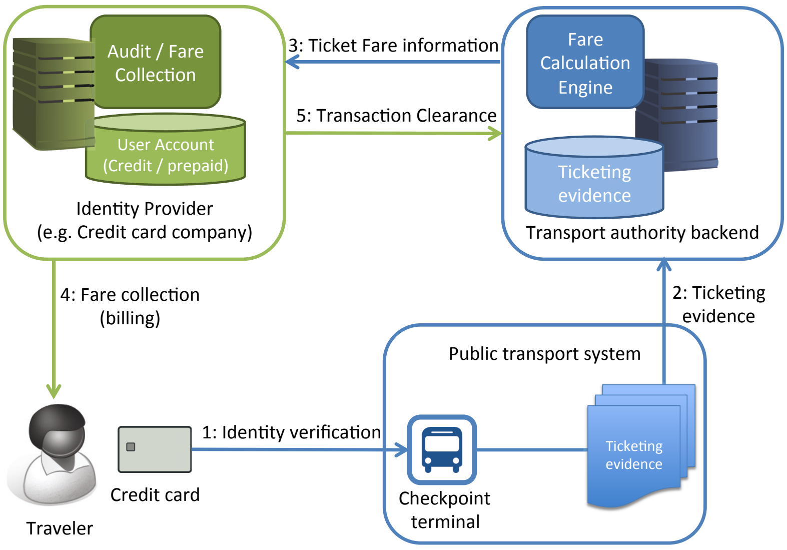

Figure 4.1. An account-based ticketing architecture with a credit card as a traveler's identity

4.1 Account-based ticketing

In account-based ticketing, each traveler is represented by a travel account maintained by an identity provider. An identity provider distributes ticketing credentials to its users who can use the credentials for accessing public transport systems. While traveling, the checkpoint terminals at transport endpoints (entries and exits) verify travelers' identities using these credentials before allowing them to enter or exit the transport system. During the identity verification process, the checkpoint terminals bind the identity of travelers with contextual information, (e.g. date, time, location information) to produce *ticketing evidence*.

From the perspective of the public transport authority, it has been identified that the cost of ticket-fare collection (i.e. operating the ticketing system and collecting money from travelers) is significant [76]. Thus, the provisioning and maintenance of ticketing credentials, as well as the fare collection, can be outsourced to third parties such as credit-card companies, telecom operators, or any other stakeholder that is prepared to take on such a responsibility [88]. The identity providers are responsible for the security of their identity verification mechanisms and carry the liability for journeys made by their customers (the travelers). However, the transport authority is responsible for gathering ticketing evidence for *ticket-fare calculations* as the travelers' journeys are intimately tied to the transport service itself.

Figure 4.1 depicts an example of an account-based ticketing architecture that allows travelers to use their credit cards for traveling. While traveling, a checkpoint terminal verifies the traveler's identity associated with her credit card (Step 1). The *transport authority backend* collects ticketing evidence (Step 2), calculates the ticket fare associated with the journey and sends the ticket fare information to the credit-card company (Step 3). Later, the credit-card company collects the fare from travelers (Step 4) based on their journeys (e.g. at the end of the month) and settles the clearance of payments with the transport authority (Step 5).

Gated and non-gated transport systems: We can categorize public transport systems into two types: *gated* and *non-gated*. In gated transport systems, the transport end-points (entries and exits) are controlled via physical gates, such as London Underground[2]. These gates are equipped with online checkpoint terminals that can verify the identity of a user and potentially consult the transport authority backend server in real-time to perform all the necessary validation on entry and exit.

In non-gated systems, there are no control gates at the transport end-points, e.g. the S-Bahn[3] railway systems in Berlin. Instead, there are usually offline, resource-constrained devices (e.g. battery operated NFC readers or contactless smart cards) that serve as checkpoint terminals. Travelers are required to tap-in/tap-out with their NFC-enabled devices at these checkpoint terminals to produce ticketing evidence. Unlike gated transport systems, travelers store the ticketing evidence, on their devices as they may be required to provide it during random ticket inspections.

A ticket verification session in gated transport systems is time bound. The time taken to verify a traveler's identity at checkpoint terminals should not exceed 300 milliseconds (ms) [2] to minimize congestion at gates. The maximum data rate that NFC hardware supports is 424 Kilo-

[2]`https://tfl.gov.uk/modes/tube/`

[3]`http://www.s-bahn-berlin.de/en`

bits per second. However, the actual data rate between NFC applications on mobile phones is usually much lower, e.g. 8 Kilobits per second or approximately 1000 bytes per second Publication II. At this rate, it takes nearly a second to transfer an X.509 certificate containing a standards-compliant RSA digital signature with a key size of 1024 bits. Therefore, it is evident that the standards-compliant RSA digital signature scheme is impractical to use as an identity-verification mechanism due to its message exchange size.

In contrast, the non-gated transport systems can tolerate longer ticket verification sessions. As the offline checkpoint terminals are inexpensive compared to the infrastructure cost of the gates and online terminals, multiple checkpoint terminals can be placed at the transport end-points to serve a large number of travelers. However, collecting ticketing evidence reliably and timeously for ticket-fare calculation from the offline checkpoint terminals is a challenge.

4.2 Research Questions

Based on Publication II, Publication III and Publication IV, this chapter investigates the research questions listed below to design identity verification schemes that can be used in account-based ticketing. The schemes are intended to work reliably in both gated and non-gated transport systems with NFC-enabled devices.

RQ 4. How to design secure identity-verification schemes that can be used for public transport ticketing over low bandwidth channels, such as NFC, which are constrained by the message exchange sizes?

RQ 5. How to reliably collect ticketing evidence produced by offline checkpoint terminals for fare calculations?

4.3 Methodology

The methodology used to answer the research questions involved is as follows:

a) designing an account-based transport ticketing architecture

b) designing various identity verification schemes
c) implementing the schemes on real devices
d) evaluating the system's security
e) developing a prototype for a user trial to evaluate usability.

The following subsections describe the adversary model for the public transport ticketing architecture and define the security requirements as well as other requirements for the correct functioning of the account-based ticketing system.

4.3.1 Adversary model

Similar to Chapter 3, there are two types of adversaries: (1) a *local adversary* and (2) a *remote adversary*. The local adversary is the one who has physical access to the user's device, such as the device user herself or an adversary who gains access to the device. The local adversary can also be someone within the vicinity of the device who can interact with the device over the NFC channel. In contrast, the remote adversary does not have physical access to the device but can execute arbitrary code on the device. However, the adversaries are incapable of influencing the operation of the TEE. The adversaries have the following main objectives:

- **Credential manipulation:** The device owner is a local adversary who intends to manipulate ticket credentials, e.g. roll back expired tickets, produce counterfeit ticketing evidence to travel free, and provide invalid ticketing evidence to pay lower travel fares.
- **Cloning credentials:** The local adversary may also intend to clone ticketing credentials from the device onto another device, or copy recent ticketing evidence to allow two or more travelers to travel simultaneously with the same credential. Similarly, malware on the device could copy and deliver ticketing credentials or ticketing evidence to the remote adversary.
- **Relay attack:** The local adversary within the vicinity of the user's device can interact with the device over NFC. The adversary can use other NFC devices to create a relay channel [26] between the user's device and the checkpoint terminal to complete a ticketing session, and then travel using the ticketing evidence.

The communication channels between the device and identity providers are assumed to be secured using standard TLS connections. However, the communication between the device and the checkpoint terminal is susceptible to relay attacks.

4.3.2 Requirements

In addition to the requirements for hosting credentials on a user's device (Section 3.3.2), identity verification schemes for the account-based ticketing architecture should fulfill the following requirements:

R4.1. **Identity verification:** Checkpoint terminals should be able to use cryptographic protocols to verify travelers' identities. Gated transport systems should not exceed identity-verification sessions beyond 300 ms in duration.

R4.2. **Generation of ticketing evidence:** The identity verification process should produce an unforgeable (by travelers or other third parties) ticketing evidence that is cryptographically bound to the identity of the traveler with the location of the checkpoint terminal. A traveler should not be allowed to generate ticketing evidence without being present at a checkpoint terminal. Entities with appropriate keys, such as the transport authority backend and personnel deployed for ticket inspection, should be able to verify the ticketing evidence cryptographically.

R4.3. **Collection of ticketing evidence:** The transport authority backend should be able to collect the ticketing evidence for ticket-fare calculation.

R4.4. **Prevent double spending:** A travel ticket cannot be used more than once. Multiple travelers should not be allowed to travel with a single piece of ticketing evidence.

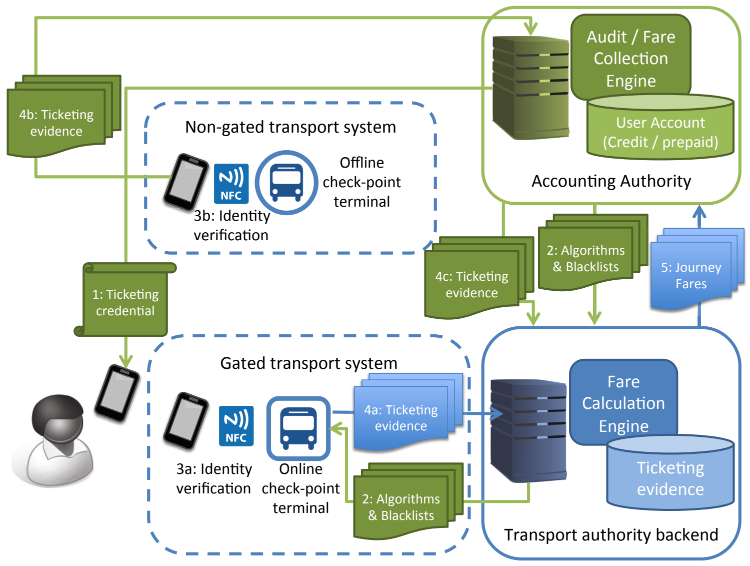
Audit / Fare Collection Engine
User Account (Credit / prepaid)
Accounting Authority
4b: Ticketing evidence
Non-gated transport system
Offline check-point terminal
3b: Identity verification
NFC
1: Ticketing credential
4c: Ticketing evidence
2: Algorithms & Blacklists
5: Journey Fares
Gated transport system
4a: Ticketing evidence
Fare Calculation Engine
Ticketing evidence
3a: Identity verification
Online check-point terminal
2: Algorithms & Blacklists
Transport authority backend

Figure 4.2. Account-based ticketing architecture for NFC-enabled device

4.4 Account-based ticketing architecture for NFC-enabled devices

4.4.1 Architecture

Figure 4.2 depicts an account-based ticketing architecture designed for public transport systems with NFC-enabled devices. The NFC-enabled device includes a *ticket application* that maintains the ticketing credentials, interacts with checkpoint terminals and provides UI to interact with a user (traveler). Similar to the virtual smart card application in Chapter 3, the ticket application includes a TA (that runs in the TEE) to protect the ticketing credentials and ticketing operations on the device.

The accounting authority provisions ticketing credentials to the devices which include ticket information and a digital certificate signed by the accounting authority (Step 1). The accounting authority is also responsible for billing, collecting ticket fares from the travelers, and clearing transactions with the transport authority. It also provides the transport authority with algorithms and secrets required to verify travelers' identities (Step 2). Additionally, the accounting authority maintains a blacklist of the devices or travelers that do not fulfill their monetary obligations or have been subjected to identity theft. The accounting authority also provides the blacklist periodically to the transport authority. The transport authority then distributes the blacklist, identity verification algorithms and secrets to online checkpoint terminals (Step 2).

The transport authority provides transport services and operates checkpoint terminals. The transport authority backend, that gathers ticketing evidence (Step 4a), calculates the ticket fare and sends requests to accounting authorities for payments collection (Step 5).

Travelers tap their NFC-enabled devices to a checkpoint terminal at a transport station to begin their journey, and complete their journey by tapping their device again to a checkpoint terminal at their destination. While tapping, the ticket application on the device and the checkpoint terminal perform an identity verification protocol. The checkpoint terminal validates ticketing credentials, identifies the traveler and cryptographically binds the location information of the transport endpoint along with the date and time of the event to produce ticketing evidence (Step 3a). The ticket application also stores the messages exchanged during the identity verification protocol as the proof of tapping (ticketing evidence) required during random ticket inspection. The ticket application later submits the ticketing evidence to the accounting authority for auditing purposes.

In the gated transport systems, the gates connected to the checkpoint terminals only allow travelers with a valid identity and ticket information to travel. Travelers listed in the blacklist are denied access. The checkpoint terminals are also responsible for generating and delivering ticketing evidence to the transport authority backend either periodically or immediately.

In the non-gated transport systems, contactless smart cards are used as the offline checkpoint terminals. Each smart card includes an application that interacts with the ticket applications on devices to produce ticketing evidence (Step 3b). Ticket applications obligate travelers to deliver the ticketing evidence to their accounting authority (Step 4b), which forwards the ticketing evidence to the transport authority backend for fare calculation (Step 4c). To enforce reporting of the ticketing evidence, the TA allows the ticket application to perform the identity verification protocol only a fixed number of times before it must submit pending ticketing evidence. When the threshold is reached, the TA ceases interacting with the checkpoint terminals until an acknowledgment is released from the transport authority.

Section 5 of Publication II presents the account-based ticketing architecture for NFC-enabled devices in detail. Similarly, Section 5 of Publication III describes the architectural extension for non-gated transport systems in detail.

4.4.2 Identity verification schemes

Publication II and Publication III explore different possible approaches in the design space to discover how we can optimize the performance to complete identity verification within the 300 ms time budget. Altogether the publications present four identity verification protocols for gated transport systems. The protocols follow challenge-response patterns during an identity verification session. During an identity verification session, a checkpoint terminal sends a challenge to the ticket application on the traveler's device. The ticket application computes a response to the challenge (inside the device TEE) using its ticketing credential and returns the response to the checkpoint terminal over NFC. Similarly, Publication III and Publication IV present an identity verification protocols designed for non-gated transport systems.

A trusted channel between an NFC modem of a device and TA would prevent malicious application on the REE from snooping the messages of identity verification protocols. However, it doesn't require a trusted UI for user input during identity verification sessions as operations on ticketing credentials activates without requiring explicit endorsements from users to minimize the session time. Trusted UI is further discussed in Section 6.3

Protocols for gated transport systems

P1: In this protocol, the ticket application generates an RSA key pair whose public key is certified by the accounting authority. The public key certificate (524 bytes) only includes essential contents such as the public key modulus of the 256 byte RSA key, 12 bytes of ticket information (i.e. the identity of the traveler and validity period) and the RSA signature (256 bytes) signed by the accounting authority. The certificate is used as part of a ticketing credential.

During identity verification, a checkpoint terminal sends a challenge to the traveler's device. The ticket application on the device computes a standard-compliant RSA signature as a response (inside the TEE) to the challenge using its private key. It then returns the response along with its public key certificate. The checkpoint terminal validates both the public certificate of the traveler and the response before allowing the traveler to enter/exit the transport system. (Section 6.1 of Publication II describes this protocol in detail.)

P2: This protocol uses a short-lived attribute certificate (valid for few hours) instead of the public key certificate as an access token. The access token (16 + 256 bytes) consists of 16 bytes of ticket information signed by the accounting authority. During identity verification, the ticket application only produces a message authentication code (MAC) (e.g. 20-byte HMAC-SHA1), in the device TEE, over the challenge using a secret key (e.g. 128-bit key) shared with the accounting authority. The checkpoint terminal only validates the access token before allowing the traveler to enter/exit the transport system. The terminal forwards the MAC to the transport authority backend server along with the ticketing evidence. The transport authority requests the accounting authority to validate the MAC during ticket-fare clearance. (Section 6.2 of Publication II describes this protocol in detail.)

P3: This protocol is similar to protocol **P2** but additionally uses a reverse hash chain. The ticket application on a traveler's device generates a reverse hash chain from a random seed in the TEE of the device. Each element in the hash chain is the hash value of its previous element starting from the seed. The last element of the hash chain is included in the attribute certificate to reduce the frequency of fetching access tokens from the accounting authority.

During identity verification, the ticket application includes the next unused element in the hash chain in reverse order, starting from the penultimate element, while computing the MAC. Similar to protocol **P2**, a checkpoint terminal only validates an access token and forwards the MAC to the transport authority backend server along with the ticketing evidence. The transport authority requests the accounting authority to verify MACs and hash chain elements during ticket fare clearance. (Section 6.3 of Publication II describes this protocol in detail.)

P4: This protocol uses the RSA signature with message recovery to minimize the size of the public key certificate (i.e. the part of a traveler's ticketing credential). Typically the RSA signature in a digital certificate is calculated over the hash of the certificate data padded using PKCS#1 with the optimal asymmetric encryption padding (OAEP) scheme [54]. However, in this protocol, the padding is replaced with much of the certificate data itself. The remaining certificate data that exceeds the padding length is symmetrically encrypted with AES in the counter mode using the hash of the certificate data itself as the 128-bit AES key.

The tailored RSA signature and the remaining encrypted data is treated as part of the ticketing credential. The checkpoint terminals require the algorithm to extract information from the tailored RSA signatures. This protocol also requires the public key of the accounting authority to be treated as a secret shared between the accounting authority and the checkpoint terminals. Section 6 of Publication III explains this scheme of generating the ticketing credential in detail.

Similar to protocol **P1**, this protocol follows the challenge-response pattern during an identity verification. The ticket application produces a standard-compliant RSA signature as the response (inside the TEE) using its private key. The checkpoint terminal validates the tailored RSA signature to extract the ticket information and the public key of the traveler's ticket application. The checkpoint terminal uses the extracted public key to verify the response before allowing the traveler to enter/exit the transport system. (Section 7 of Publication III describes this protocol in detail.)

Protocols for non-gated transport systems

Unlike online readers in gated transport systems, the checkpoint terminals in non-gated transport systems use limited processing and limited memory devices such as smartcards. Therefore, the protocols are designed in such a way that the checkpoint terminals need not require to receive and verify the traveler's ticketing credentials. Instead, the protocols rely on the device-specific counters binding between the traveler's device and the checkpoint terminal.

P5: In this protocol, both the ticket application and the offline checkpoint terminal cryptographically bind their device-specific counters and their public key certificates to the ticketing evidence. The ticket application and the offline checkpoint terminal share their respective secret keys with the accounting authority, which they used to attest their device-specific counters by generating signatures with the keys. The ticket application prepares the device for the next identity verification session immediately after the previous session. The TA of the ticketing application signs its TEE counter states using the secret key shared with the accounting authority as the commitment of the counter value.

During identity-verification, the ticket application sends the commitment of its counter value as a challenge to the checkpoint terminal. The checkpoint terminal signs the challenge along with its device-specific counter. Then, it returns the response along with its device certificate. The TA then cryptographically binds the identity of the TEE and the current state of its counter with the identity and the counter state of the checkpoint terminal in an unforgeable manner to produce ticketing evidence. Later the ticket application submits the ticketing evidence to the accounting authority (Section 3 of Publication IV provides further detail).

To enforce reporting of the ticketing evidence, the TA ceases producing the commitments for its counter value after a limited number of interactions with checkpoint terminals. After which, the traveler must submit her pending ticketing evidence to the transport authority in order to receive an acknowledgment that allows the TA to continue to produce commitments for the counter value.

P6: This protocol is an extension to protocol **P5** for supporting devices where TEEs are unavailable or practically inaccessible. The ticket application on the device must, therefore, fetch a challenge as an access token (tokenized challenge) from the accounting authority before the identity verification. The ticket application is also required to submit the ticketing evidence before it can receive a new challenge (Section 5 of Publication IV describes this protocol in detail.)

4.4.3 Implementation

The ticket application for the protocols for gated transport systems (protocols **P1**, to **P4**) was implemented on Nokia C7 smartphones running Symbianˆ3. The application included a TA that ran on On-board Credentials (ObC), the TEE available on that smartphone platform. An NFC reader (ACR 122U) connected to a Linux PC running Ubuntu 10.10 was used as a checkpoint terminal. The NFC reader was configured to operate at 106 Kilobits per second. Although NFC-enabled devices can support higher data rates, the protocols were designed to support for real devices with lower data rates.

The ticket application with protocol **P5** was initially implemented on a Nokia C7 smartphone running Symbianˆ3. The protocol was also integrated into the Nokia public transport app on the Nokia 603 smartphone. SmartMX tags from NXP were used as offline checkpoint terminals, and the terminal application was developed using Javacard 2.2.1. A public trial along the Port Washington Branch of the Long Island Railway Road operated by New York Metropolitan Transit Authority[4] was conducted with over 100 participants. The Nokia X6 smartphones were distributed to trial participants. A user study conducted at the end of the trial gave very encouraging feedback and provided the motivation to continue the exploration of NFC-based ticketing in general. The study found that 42% of the participants preferred the ticketing credentials on smartphones compared to only 24% for a smart card or payment card. The trial does not consider the demographic distribution among the participants. However, future reporting on trail should consider such information.

The ticket application for protocol **P6** was implemented on a Google Nexus S smartphone. Section 7 of Publication II, Section 9 of Publication III and Section 6 of Publication IV provide details on the protocol implementation and performance measurements. Section 8 of Publication II and Section 10 of Publication III provide system analysis in detail.

[4]`http://www.mta.info/lirr`

Table 4.1. Identity verification time for different protocols in milliseconds (ms).

Protocol	Response	Access token	Verification at checkpoint	Session time
P1	RSA signature	Optimized certificate	Access token, Response	482 ms
P2	MAC	Attribute certificate	Access token	228 ms
P3	MAC + Hash element	Attribute certificate	Access token	246 ms
P4	RSA signature	RSA signature with message recovery	Access token, Response	256 ms
P5	counter commitment of terminal	TEE counter commitment	not essential	305 ms
P6	counter commitment of terminal	Tokenized challenge	not essential	820 ms

4.5 Discussion

Identity verification session time: All the protocols described in this chapters use digital certificates to prove the identity of a traveler. The certificates can be cryptographically verified using the public key of the accounting authority (Requirement **R4.1**).

Table 4.1 shows the time taken for identity verification using different protocols. The table also shows the content of response messages for each protocol and messages that are verified at the checkpoint terminals. With the exception of protocol **P1**, all other protocols for gated transport systems completed the identity verification within the allowable time (i.e. 300 milliseconds) for considerable key size (larger than 1024 bits). As explained above, the identity verification for non-gated transport system are not strictly time bound. Space-efficient algorithms, such as elliptic curve cryptography, can further reduce the time and also use stronger keys. However, NFC readers deployed in public transport systems during the protocol development only included hardware acceleration for the RSA algorithm. Therefore designing protocols with other algorithms would have require costly hardware upgrades. SmartMX tags used in Publication V required longer time to produce commitments for its counter values in protocol **P6** (with Google Nexus S) than in protocol **P5** (with Nokia 603). It is highly likely that the exact alignment of the antennas in a device used had significant impact on the performance.

Ticketing evidence: The protocols cryptographically bind the identities of travelers with the time and location information of the corresponding checkpoint terminals using a challenge-response pattern. The responses are generated in the TEE of travelers' devices using keys protected by TEEs. A response is either a digital signature produced using the private key of a traveler, or a MAC generated using a secret key shared with the accounting authority. The protocol messages are used as the ticketing evidence which can be verified with appropriate keys (Requirement **R4.2**).

Protection against double spending: The protocols use one of three possible techniques to prevent double spending: (a) verifying responses at the checkpoint terminals, or (b) binding TEE-specific counters into ticketing evidence, or (c) using time-windows for access tokens. (Requirement **R4.4**)

Collection of ticketing evidence: In gated transport systems, checkpoint terminals deliver the ticketing evidence to the transport authority backend. In contrast, in non-gated systems, travelers are responsible for reporting their ticketing evidence to the accounting authority, which forwards the ticketing evidence to the transport authority (Requirement **R4.3**).

Privacy concerns: One primary concern of this architecture is protecting travelers' data privacy from the transport authority and the accounting authority. These authorities can create a journey profile of travelers based on their travel history. However, such route information could legitimately be used to suggest alternative routes to the travelers during traffic congestion. Further, it could also be used to implement flexible pricing policies such as congestion pricing, and transfer discounts.

However, the distribution of blacklisted identities to a transport authority can have serious privacy concern. For example, the transport authority can identify travelers with bad credit histories. Therefore, identity providers can issue randomized identities each time they re-issue ticketing credentials to prevent the transport authorities from identifying travelers after the credentials have been re-issued. Issuing a new identity will ensure that a traveler with her identity blacklisted will not reuse the same identity after she has reconciled with her identity provider. To prevent authorities from profiling travelers based on their travel history, we can design the transport authority backend to utilize TEE-enabled servers. The TA running in the server's TEE would share unique keys with TEEs of travelers' devices and with checkpoint terminals separately. The shared keys would then be used to encrypt the ticketing evidence. Similarly, ticket fare calculation can be implemented as a TA to prevent servers from learning travelers' profiles. Privacy preserving computation on server infrastructure using a TEE is discussed in the next chapter.

4.6 Conclusions

Identity verification schemes described in this chapter use cryptographic protocols to verify the identities of travelers securely, to extract journey information for fare calculations and to prevent ticket fraud. Furthermore, the protocols for gated transport systems are optimized to complete within the allowable time. Therefore, these protocols can be used as a secure public transport ticketing over low bandwidth channels such as NFC (**RQ 4**).

The protocols for non-gated transport systems use devices as a medium for collecting ticketing evidence from offline checkpoint terminals. Further, the protocols obligate devices to report the ticketing evidence by restricting the number of times a ticket can be used before reporting the evidence (**RQ 5**). Nearly twice as many participants in the user study preferred using smartphones over smart cards for public transport ticketing.

5. Privacy preserving computation on server infrastructure

Chapter 3 and Chapter 4 described how to use TEEs on users' devices to establish the level of trust required by credential issuers and service providers for provisioning, hosting, and using credentials. This chapter describes how to establish trust in the server infrastructure, such as cloud servers, using a TEE to provide privacy guarantees. In particular, this chapter presents the design and evaluation of a privacy-preserving membership test using a TEE on a server. One use case for this is to provide cloud-based mobile malware checking services.

The recent evolution of cloud computing technology has promoted migration of many local as well as remote services to cloud platforms. One such service is malware checking, which used to operate as a local service on devices, but has moved to a cloud-assisted setting. For example, Google's Verify Apps for Android is a cloud-assisted service that notifies users when they intend to install applications that are potentially dangerous [41]. However, as explained in Section 1.1.2, data privacy is a major concern in cloud computing.

In the cloud-assisted design, anti-malware vendors host their malware databases on the cloud and run malware checking services from the cloud. Users may query the service with application identifiers to learn if an application is a potential malware. If a user receives a positive indication from the service, she may further consult the anti-malware vendor directly to receive instructions on how to deal with the malicious application. However, for privacy reasons users do not wish to reveal which applications they have installed. It has been shown that a single snapshot of applications installed on a device can reveal personal characteristics of the user, such as their gender, religion, relationship status and interest. [86]. This information can be used to profile users. Therefore, we need a mechanism such as a private membership test (PMT) for hosting privacy-sensitive services in the cloud.

5.1 Private membership test

Formally in a PMT, a server known as a *lookup server* holds a large data set of n entries called a *dictionary* such that $X = \{x_1, x_2, ..., x_n\}$. A user runs a PMT protocol with the lookup server to know if an entry q is a member of X. The output r of the PMT protocol is a single bit indicating the membership status of q, i.e. r = 1 if $q \in X$ and 0 otherwise, without revealing q and r to the lookup server.

PMT using cryptographic primitives: A PMT protocol can be built using cryptographic primitives such as *private set intersection (PSI)* [77, 78] or homomorphic encryption [68]. However, these schemes are expensive in terms of computation and communication. For example, to complete a PSI on a dictionary of size n requires $\mathcal{O}(n)$ communication between a user and a server, and also requires the server to perform $\mathcal{O}(n)$ operations. In addition, cloud-based malware checking services are expected to handle simultaneous lookup queries from a large number of users. However, in these schemes the number of simultaneous queries that can be supported is small, thereby affecting server-side scalability of the schemes.

PMT using TEEs: Several prior works [84, 9] have used TEEs to establish trust in cloud platforms. TEEs allow them to guarantee confidentiality and integrity of the applications and data hosted in the cloud. A straight-forward approach for designing a cloud-based PMT protocol using a TEE is to integrate a TEE on the lookup server as depicted in Figure 5.1.

The *dictionary provider* generates a dictionary and hosts it on the lookup server. A TA running in the TEE provides the actual lookup functionality. On the other hand, the REE application called *client app* facilitates interaction between users and the TA and makes the dictionary accessible to the TA. If the lookup functionality were not performed in the TA, then a malicious lookup server operator can modify the REE of the lookup server to learn the content of the users' queries and their results.

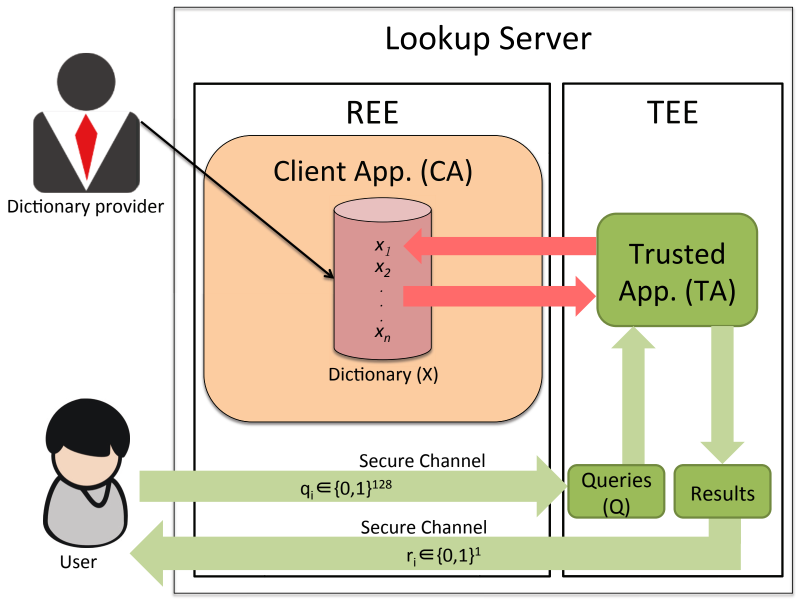

Figure 5.1. A cloud-based private membership test

Users can establish secure communication channels with the TA and submit their queries directly to the TA. The TA then performs a membership test on the dictionary and returns the results to the users. The secure channel prevents the lookup server operator from learning the content of the queries and their results.

The dictionary is often too large to fit entirely in the TA, and therefore must be stored in the REE. Therefore, an adversary who controls the REE can observe the dictionary access pattern and possibly infer information about the queries.

Technologies such as *Oblivious RAM* (ORAM) [39] can be used to prevent information leakage through memory access patterns. We can implement the state-of-the-art in ORAM technology *Path ORAM* [90] as a TA to obscure the dictionary access patterns in the PMT protocol and prevent the adversary from learning information about the queries. Path ORAM has a constant communication overhead and only $\mathcal{O}(\log n)$ computational cost per query on a dictionary of size n entries. However, it suffers from poor scalability, i.e. supporting simultaneous queries from m users will incur $\mathcal{O}(m \log n)$ cost.

5.2 Research Question

Based on Publication VI, this chapter investigates the following research question.

RQ 6. Can we design a privacy-preserving PMT service using a TEE (available on the server) that exhibits better scalability than ORAM by supporting a higher number of simultaneous queries?

5.3 Methodology

The methodology used to answer the above research question involved the following actions:

a) designing a lookup server architecture for the cloud-based mobile malware checking scenario
b) designing a PMT using a TEE on the lookup server, which allows simulations query lookups whilst also protects users privacy
c) implementing the architecture on two different real TEE platforms
d) measuring the individual query response latency and the overall query throughput, in order to compare the performance against the ORAM-based approach.

The following subsections describe the adversary model for the PMT protocol and define the requirements for this service.

5.3.1 Adversary model

Two types of adversaries are considered for the PMT protocol: (1) a malicious *lookup server operator*, and (2) the *dictionary provider*. The malicious operator of the lookup server is the primary adversary. It has full control of the lookup server's REE. The dictionary provider is only a secondary adversary who is assumed to provide legitimate malware identifiers in the dictionary. As discussed above, it is acceptable for the dictionary provider to have access to the queries that are revealed to it by the users.

The primary intention of these adversaries is to profile users based on their queries. The dictionary provider can only profile users based on the queries revealed to it directly by users. However, the malicious lookup server operator can perform side-channel attacks to learn users' queries. The lookup server operator is assumed to be incapable of performing hardware-level attacks, as the cost of performing such attacks is comparatively more expensive than the value of the data it can extract. Therefore, the adversary cannot observe or influence the internal state of TA and its operations in its private memory. However, the adversary can attempt to infer memory accesses and internal operations of the TA using software-based side-channel attacks. The adversary can also observe and measure the duration of interaction by the TA with non-private memory, count the number of queries going into TA, measure individual query response latencies and submit arbitrary queries.

The lookup server operator can attempt to modify the dictionary. To protect against dictionary modification, the dictionary provider can attach a digital signature or a MAC (computed using a key shared between the dictionary provider and the TA), which the TA can use to verify the authenticity of the dictionary. The dictionary provider is assumed to be an honest-but-curious adversary, which is not expected to subvert the dictionary. The communication channels between the users' devices and the TA are assumed to be secured using correctly implemented cryptographic primitives, e.g. using a standard TLS connection. However, since the adversary controls the REE, it can arbitrarily schedule and/or drop encrypted messages exchanged between users and the TA.

5.3.2 Requirements

The following requirements apply to the cloud-based PMT protocol:

R5.1. **Query privacy:** The lookup server operator must not be able to learn anything about the content of the users' queries or their responses. However, the dictionary provider may learn the content of the queries that users reveal to it directly.

R5.2. **Support for simultaneous queries:** The PMT protocol should be able to handle simultaneous queries from a large number of users (e.g. in the order of thousands of queries per second) without a deterioration in the level of overall query processing throughput.

Section 4 of Publication VI provides further requirements to ensure security, performance, and accuracy of the cloud-based malware checking service using the PMT protocol.

5.4 Privacy-preserving PMT design using a TEE

5.4.1 The carousel approach

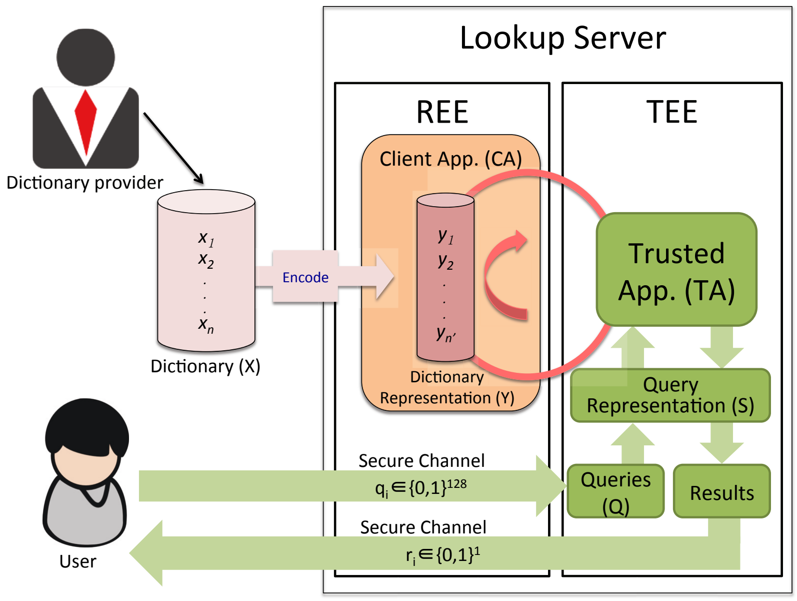

Figure 5.2. A Privacy-preserving PMT architecture with carousel design pattern

To prevent information leakage during dictionary access, Publication VI introduces a new *carousel design pattern* in which the entire dictionary is continuously circled through the TA. Figure 5.2 depicts the privacy-preserving PMT architecture with carousel design pattern.

The PMT architecture is designed such that it prevents the TA from responding to a query until the query has completed one full carousel cycle in the TA. Not answering a query immediately prevents the adversary in the REE from learning the position of the query in the dictionary. Since the TA only responds to queries that have completed a full carousel cycle, two factors influence the time taken to respond (i.e. response latency): *dictionary size* and the *processing efficiency*.

One key characteristic of the mobile malware checking scenario is its tolerance on a *non-zero false positive rate* (FPR). A study by Yahoo [103] found that typical Android devices have around 95 installed apps. Also, a leading anti-malware vendor [55] indicated that an FPR of 2^{-10} is acceptable for cloud-based mobile malware checking. With such a non-zero FPR, the majority of users should not encounter more than one false positive during the lifetime of their devices. Therefore, a typical user would disclose at most one application identifier to the dictionary provider, thereby virtually eliminating the privacy concern. Such non-zero FPR allows the use of data structures such as Bloom Filters and Cuckoo hash for an efficient membership test. These data structures allow the transformation of the dictionary into a more compact format, reducing the amount of data that needs to be cycled through the TA.

Dictionary Representation

The dictionary provider transforms the dictionary X into *dictionary representation* Y, which is then cycled through the TA. The performance of the system depends on the choice of the data structure used for generating the dictionary representation. Publication VI explores the following data structures to transform a dictionary into its representations for efficient membership tests, and compares the response latencies of each representation.

1. *Sequence of Differences:* Y is generated by representing only the difference between successive entries in X. When inserting an entry x_i into Y, x_i is first hashed to a value of fixed sized length ($\varepsilon + log\ n$ bits), where ε is based on the FPR value, i.e. $FPR = 2^{-\varepsilon}$, and n is the number of entries in X. The hash values are sorted, and the differences between each successive hash values make up Y. When multiple entries result in the same hash value, only one copy of the hash value is added in Y to avoid leaking information. Since the size of the differences is smaller than the size of the hash values themselves, we can reduce the size of Y significantly. During carousel processing, the sequence of differences is used to reconstruct the hash values. During the membership test for an entry q_i, the hash value s_i of q_i is

matched against each hash value reconstructed from the sequence of differences. Section 6.2 of Publication VI describes how to use the sequence of differences to generate Y from X in detail.

2. *Bloom Filter:* A Bloom filter is a bit array of length N bits initialized with 0s. It consists of l hash functions whose output is uniformly distributed over a domain of size [0, $N - 1$] [11]. The FPR value determines the number of hash functions and the resulting size of the Bloom filter, which is approximately $1.44\varepsilon n$ bits. When inserting x_i in Y, the corresponding Bloom filter positions of x_i are calculated by applying the l hash functions to it. After which, the bit values at the corresponding Bloom filter positions in Y are set. During the membership test for q_i, the l hash functions are applied to q_i, and the bit values at its corresponding Bloom filter positions in Y are checked. If all the bits at the positions are set, then q_i is a probable member of Y. Section 6.3 of Publication VI describes how to use a Bloom filter to generate Y from X in detail.

3. *4-ary Cuckoo Hash: cuckoo hash* [73] is another data structure that can be used for efficient membership tests. A variant of cuckoo hash, called *4-ary cuckoo hash* with four hash functions was selected as it utilizes approximately 97% of the hash table [23]. When inserting x_i into Y, all four candidate positions of x_i are calculated by applying the four hash functions to it. After which, the value of x_i is truncated to a (ε+2)-bit value, say y_i, which is then inserted into the first available candidate position in Y. If all four positions for y_i are occupied (say, by values $\hat{y_1}, \hat{y_2}, \hat{y_3}, \hat{y_4}$), then y_i is inserted into one of the positions by recursively relocating $\hat{y_j}$ into one of its three remaining candidate positions. During the membership test for q_i, the value of q_i is first truncated to a (ε+2)-bit value s_i, then the four candidate positions for q_i is calculated. After which, the values at the positions in Y are checked against s_i to determine if q_i is a member of Y. Section 6.4 of Publication VI describes how to use a 4-ary cuckoo hash to generate Y from X in detail.

Carousel processing

The *client app* on the lookup server divides Y into several chunks and passes each chunk sequentially to the TA along with waiting queries. Passing Y as a sequence of chunks allows the TA to minimize the waiting time for an arriving query before the TA can start processing it (i.e. when the TA receives the next chunk). The TA transforms the queries into their representations in a similar manner to an entry in Y and maintains the representations in a sorted list S for processing. It compares each entry in the chunk with S and records the results. This process is repeated for each chunk. While processing Y, the TA performs a constant number of operations per entry independent of the size of S. Section 5 of Publication VI provides the complete description of the carousel processing.

5.4.2 Implementation

At present, the number of Android malware samples is in the order of millions whilst the number of new malware samples is growing roughly linearly. In 2015, G Data, an anti-malware vendor, reported approximately 2.3 million new Android malware samples, with this a 50 % increase on the previous year. Therefore, a dictionary of 2^{26} entries (i.e. ~67 million entries) was chosen with an estimation that allow the PMT protocol to support malware checking for the next ten years.

The lookup server using the carousel approach was implemented on the two most prominent hardware security architectures currently available: ARM TrustZone and Intel SGX. The dictionary representations described above were used with $\varepsilon = 10$ on ARM TrustZone and $\varepsilon = 14$ on Intel SGX.

Kinibi on ARM TrustZone

For the ARM TrustZone-based implementation, a Samsung Exynos 5250 development board from Arndale[1] with a 1.7 GHz dual-core ARM Cortex-A17 processor was used. The board runs Android OS (version 4.2.1) as the primary OS on the REE and Kinibi OS as the secure OS in the TEE. Kinibi allows authorized TAs to execute inside the TEE.

Kinibi on the development board only allows a TA to use approximately 900 kB of *private memory* for heap and stack data structures. The limited size of the private memory sets the boundary for the number of queries

[1]`http://arndaleboard.org/`

that the TA can process concurrently. Additionally, Kinibi only allows the client app to share 1 MB of additional memory with the TA. This shared memory is used for transferring chunks of Y and queries to the TA, and for retrieving responses. Sections 7.1 and 7.2 of Publication VI provide further details of the lookup server implementation on Kinibi.

Intel SGX

For the Intel SGX-based implementation, an SGX-enabled HP EliteDesk 800 G2 desktop PC with a 3.2 GHz Intel Core i5 6500 CPU and 8 GB of RAM was used. The desktop PC runs Windows 7 (64 bit version) as the host OS, with a page size of 4 kB.

Unlike Kinibi, Intel SGX does not provide the TA with *private memory* and thus the adversary can observe the TA's memory access pattern at page-level granularity (Section 2.2.2). To conceal the memory access pattern beyond the page-level, the TA is designed such that it performs the same number of memory access operations on all pages whenever a private data structure spans beyond a page. Sections 7.1 and 7.2 of Publication VI provide further details of the lookup server implementation on Intel SGX.

5.4.3 Performance evaluation against ORAM

The response latency was measured under various query loads using the two following distinct query processing patterns, which are described below.

Batch performance

Queries were sent in a batch at the beginning of each carousel cycle and the total processing time for a single batch of queries was measured. Section 7.4 of Publication VI provides the performance evaluation of carousel approaches in detail. The results show that the carousel approach with the 4-ary cuckoo hash dictionary representation, i.e. *cuckoo-on-a-carousel* (CoaC), can handle more queries with a smaller overhead than other methods. Therefore, for a comparison of the carousel approach with the ORAM-based approach, the 4-ary cuckoo hash data structure was also used for Path ORAM. Section 7.3 of Publication VI details the implementation of 4-ary Cuckoo hash on Path ORAM, *cuckoo-on-ORAM* (CoO).

Table 5.1. Total processing time for a batch of queries

Approach	**Kinibi**		**Intel SGX**	
	1 query	2000 queries	1 query	2000 queries
CoaC	0.73 sec	1.83 sec	0.109 sec	0.282 sec
CoO	0.009 sec	18.63 sec	0.001 sec	1.329 sec

CoO provides a very fast response latency for a single query compared to CoaC. However, queries are processed sequentially. Table 5.1 provides the comparison of CoaC and CoO on both TEE platforms with query load of 1 query and 2000 queries. Figures 3 and 4 of Publication VI provide a further comparison of the carousel approach with other dictionary representations in relation to CoO on both TEE platforms. The results show that the carousel schemes provide lower query response latencies when handling batches of queries.

Steady-state performance

Table 5.2. Breakdown point for CoaC and CoO on TEE hardware platforms

Approach	**Kinibi**	**Intel SGX**
CoaC	1025 queries/sec	3720 queries/sec
CoO	111 queries/sec	1354 queries/sec

In addition to the measurements of the batch processing performance, the steady-state performance of CoaC and CoO was also measured, assuming a constant query arrival rate. On both TEE platforms, the response latencies of CoaC and CoO were measured with different query arrival rates to identify the maximum query arrival rate at which the response latency could be guaranteed. This maximum query arrival rate is termed as the *break-down point* beyond which the rate of query arrival exceeds the speed of query processing, thus a queue of queries begins to build up and the response latency increases (indefinitely). At this point, the PMT service must add new hardware to guarantee response latency. Table 5.2 lists the breakdown point for CoaC and CoO on both TEE platforms. The table shows that on both platforms CoO reaches its breakdown point under a smaller query arrival rate compared to CoaC.

5.5 Discussion

Query privacy: carousel processing ensures query privacy by, (a) performing the same number of operations on every entry in Y independent of the number of queries and the content of the queries, and (b) ensuring that every query remains in the TA for exactly one full carousel cycle (Requirement **R5.1**). Section 7 of Publication VI provide implementation details avoiding information leakage during carousel processing. Section 9 of Publication VI provides further privacy analysis of PMT with the carousel approach. It also discuss on preventing side-channel attacks on SGX to prevent adversary from inferring users' query contents.

Simultaneous queries: The carousel approach supports simultaneous queries (Requirement **R5.2**) by, (a) passing the incoming queries along with each chunk of dictionary representation to TA, and (b) maintaining the query representations in sorted order. The former allows queries to be processed immediately without having them to wait for the carousel cycle to complete. The latter allows the TA to perform membership tests for queries while processing the relevant chunk of dictionary representation.

5.6 Conclusions

For the chosen dictionary size (2^{26} entries), the results discussed in Section 5.4.3 confirm that CoaC can sustain significantly higher query arrival rates. Therefore the carousel approach can be applied to designing a privacy-preserving membership test that is more efficient than ORAM while handling simultaneous queries from a large number of users (Research question **RQ 6**).

6. Discussion and Conclusion

Based on the work presented in the previous chapters, this chapter discusses five themes relevant to the use of TEEs in securing systems:

- Non-programmable vs. programmable trusted hardware
- Credential management on multiple devices
- Trusted Path
- Service augmentation with a combination of TEEs on devices and server infrastructure
- Formal analysis of protocols

6.1 Non-programmable vs. programmable trusted hardware

We can divide trusted hardware into two categories based on the functionality offered to developers: *non-programmable* and *programmable* trusted hardware. The following subsections discuss the functionality and interfaces that these trusted hardware categories offer, the different use of these types of trusted hardware, and the advantages and the disadvantages of each category.

6.1.1 Non-programmable trusted hardware

Non-programmable trusted hardware, such as TPMs, provide a predefined set of functions to the developers.
TPM: TPM specifications [95, 96] define a set of interfaces (APIs) for accessing TPM functionality from platform OSs and other applications. The APIs allow some level of customization such as generating/setting cryptographic keys, setting passwords and defining authorization poli-

cies on using the keys. However, the APIs do not permit developers to run application-specific logic for extending the functionality beyond TPM specifications.

Hardware-backed keystore: Device OS platforms, such as Android and iOS, provide a hardware-backed keystore to protect cryptographic keys and operations. Android Keystore APIs [40] allow the generation, storage and usage of cryptographic keys protected in trusted hardware. Android also allows the keystore to attest to external entities that specific keys are protected by a hardware-backed keystore, using a certificate chain where the root key is certified by Google's attestation root key.[1] The Android Keystore is only customizable in the sense that it allows developers to define the following authorization policies on cryptographic keys:

- *Key usage* policy specifies the operations that a cryptographic key is allowed to perform. For example, a key defined as a signature-only key can be used for signing but not for encryption.
- *Temporal validity interval* associated with a key that specifies the duration or period in which the key is valid for use.
- *User authentication* policy requires user verification through the Android secure lock screen credentials (e.g., pattern, PIN, password, or fingerprint) before the key can be used.

Similarly, iOS utilizes a TEE to implement a *Secure Enclave* feature that offers hardware-assisted cryptographic operations and secure key management to developers [4].

6.1.2 Programmable trusted hardware

In contrast, programmable trusted hardware allows developers to execute their application-specific logic in the trusted hardware. It can allow developers to extend the functionality of the trusted hardware through a TA and define customized interfaces for accessing the extended functionality from REE applications. There are various types of programmable trusted hardware available, such as Universal Integrated Circuit Card (UICC) and CPU-assisted TEEs.

[1] Key Attestation: `https://developer.android.com/training/articles/security-key-attestation.html`

Universal Integrated Circuit Card (UICC): The Universal Integrated Circuit Card (UICC), is widely used in mobile networks as SIM cards. UICCs are based on Java Card technology [72] and implement the GlobalPlatform card specifications [36] for loading and managing applications on cards. Therefore, UICCs allow developers to provision arbitrary applications compiled as Java Card applets on a single card. However, mobile network operators (MNOs) often regulate access to SIM cards and prohibit developers from provisioning arbitrary applications to a SIM card.
CPU-assisted TEEs: GlobalPlatform defines specifications for TEEs. The device specifications of the GlobalPlatform [31] consist of a list of comprehensive TEE specifications including interfaces for implementing a TA (TEE Internal Core API), interfaces for REE applications to communicate with a TA (TEE Client API) and a framework for managing the TA lifecycle (TEE Management Framework). Modern CPU architectures such as ARM TrustZone [7] and Intel SGX [67] augment the main CPU with system-wide TEE functionality for applications running on the system.
ARM TrustZone: The interfaces for REE applications to communicate with a TA in ARM TrustZone are not disclosed publicly, which prevents developers from implementing application-specific functions in TrustZone TEEs [99]. However, several TEE providers such as ObC [59], Kinibi [97] and OP-TEE[2] run a secure OS in TrustZone TEEs to provide interfaces for TA development as well as for REE applications to communicate with a TA. OP-TEE is a GlobalPlatform compliant TEE in the sense that it follows GlobalPlatform specifications for TA development and communicating with a TA from REE applications. In contrast, ObC provides proprietary APIs for TA development and communication. Kinibi, on the other hand, supports GlobalPlatform compliant TEE APIs in addition to its proprietary TEE APIs.
Intel SGX: Unlike TrustZone, Intel SGX does not require a separate TA to protect security-sensitive application logic. Instead, Intel SGX allows individual applications to demarcate selected code and data as *enclaves* that run in their own TEEs. Intel also provides an SDK that includes APIs, libraries, documentation, and tools for application development.[3]

[2]OP-TEE `https://www.op-tee.org/`

[3]Intel Software Guard Extensions (Intel SGX) SDK `https://software.intel.com/en-us/sgx-sdk`

6.1.3 Use of non-programmable vs. programmable trusted hardware

Table 6.1. Indication of which types of trusted hardware can be used to implement the various protocols and applications presented in this dissertation

Applications	**Trusted hardware functionality**	**Keystore APIs**	**TPM 1.2**	**TPM 2.0**	**TEE**
Hardware-supported Virtual Smart Card (Sec: 3.4.1)	Secure keystore	✓	✓	✓	✓
Public Transport Ticketing (**P1**)	Secure keystore	✓	✓	✓	✓
Public Transport Ticketing (**P2**)	Secure keystore	✓	✓	✓	✓
Public Transport Ticketing (**P3**)	Secure keystore, Hash chain	×	×	×	✓
Public Transport Ticketing (**P4**)	Secure keystore	✓	✓	✓	✓
Public Transport Ticketing (**P5**)	Secure keystore, counters, policies	×	×	✓	✓
Public Transport Ticketing (**P6**)	not essential	—	—	—	✓
Cloud-assisted malware checking (Sec: 5.4.1)	Private Membership Test	×	×	×	✓

The work presented in this dissertation utilizes programmable trusted hardware, such as TEEs, to execute arbitrary security-sensitive application logic in the form of TAs. However, we can implement many of the applications using non-programmable trusted hardware, such as TPM 2.0 or a hardware-backed keystore. For example, Nyman et al. [70] presented a design of an eID architecture based on TPM 2.0 specifications.

Table 6.1 shows which of the protocols and applications presented in this dissertation that could be implemented using non-programmable trusted hardware. The table also lists the security functions that trusted hardware offers today. However, the public transport ticketing with protocol **P6** (in Publication IV, Chapter4) does not rely on trusted hardware. It is designed for devices without trusted hardware, where the devices receive access tokens directly from the accounting authority for identity verification.

We can categorize applications based on the security functionality they utilize from a trusted hardware.

- **Secure keystore:** We can use a non-programmable trusted hardware to develop applications that require protecting cryptographic keys and use standard cryptographic operations. For example, eID credentials (in Publication I and Publication V), public transport ticketing with protocols **P1**, **P2** (in Publication I), and public transport ticketing with protocol **P4** (in Publication III) can utilize non-programmable trusted hardware to protect private keys and produce RSA signatures.
- **Secure keystore and monotonic counters:** We can use TPM 1.2 or higher to develop applications that require secure counters in addition to a secure key storage.
- **Policies on keys and monotonic counters:** We will need TPM 2.0 to implement applications that define authentication policies for using cryptographic keys and secure counters. For example, public transport ticketing with protocol **P5** (in Publication III) can utilize TPM 2.0 to limit the number of device-specific counters before a traveler submits pending ticketing evidence to the accounting authority. Protocol **P5** uses device-specific counters during identity verification before allowing travelers to travel.
- **Other functionality:** At present, we will need a programmable trusted hardware to support functionality beyond securing cryptographic keys, monotonic counters and associating security policies on the keys and counters

We may be able to implement protocol **P3** using a non-programmable trusted hardware provided that the trusted hardware facilitates secure storage of a reverse hash chain and allows REE applications to extract one element of the hash chain in reverse order. However, supporting a full PMT primitive (Sec: 5.4.1) on non-programmable trusted hardware is not possible as the nature of the membership test and the PMT dictionary depends on its applications.

6.1.4 Advantages and disadvantages

Non-programmable trusted hardware APIs are well documented, standardized, extensively analyzed and regularly maintained. Nevertheless, adding new functions and interfaces to non-programmable trusted hardware is either not possible or takes a long time. Only functions that are relevant to the majority of applications are added. However, we cannot predict the security requirements of different applications. Thus it becomes difficult to include all the non-standard cryptographic protocols and algorithms used by applications into non-programmable trusted hardware. TPM 2.0 does support algorithm agility, but this only allows manufacturers to implement any cryptographic algorithm from the list approved by the TCG [96].

On the other hand, customizable trusted hardware gives liberty to extend the functionality of the hardware with developer-defined functions and APIs. However, the extended functions and APIs are usually not audited to the same extent as non-programmable trusted hardware. Thus, any incorrect implementation may open up vulnerabilities that can compromise the security of applications. Also, preventing side-channel attacks against developer-defined functions can become challenging (as explained in Section 9 of Publication VI).

6.1.5 Future directions

Over time new functionality will be added to trusted hardware. One of the important features, the application-specific secure counter would be beneficial for developers. Intel SGX already provides APIs for accessing secure, monotonic counters [44]. We can anticipate that other trusted hardware providers would utilize the Replay Protected Memory Block (RPMB) [52] feature of an embedded multimedia card (eMMC) to facilitate application-specific secure counters. A feature to support cryptographic key migration between trusted hardware would also be beneficial as we are using more than one computing device simultaneously. Section 6.2 discusses various aspects of credential management on multiple devices in detail.

In future, we can expect new specifications that allow the accessing of trusted hardware functionality from a wide range of services. For example, GlobalPlatform has been working on APIs for enabling web applications to access SEs on devices.[4] Also recently, an IETF draft Open Trust protocol (OTrP) has been published which defines an open, interoperable protocol for trusted service managers (TSMs) to manage security domains and applications in different TEEs on various devices. We can also expect the standardization of hardware security architectures designed for resource-constrained embedded devices, such as Intel's TrustLite[5] and ARM's TrustZone-M[6], and the publication of standardized APIs to access these hardware.

6.2 Credential management on multiple devices

In today's ubiquitous computing environment, most of us own and use more than one computing device. Also, switching between devices has become the norm. Therefore from a user's perspective, it is desirable to be able to use credentials from any of her devices. Furthermore, users may also need to migrate credentials from one device to another, e.g. during device replacements.

To transfer (i.e. migrate/share) credentials protected with trusted hardware such as a TEE, we must ensure that the credentials are always transferred to compatible trusted hardware on a target device. Most of the TEEs on devices include a device-specific certificate, signed by the device manufacturer, that asserts the security level of the TEE. A straightforward approach for transferring TEE-protected credentials between two devices, say a source S and a target T, would be to first transfer the T's device certificate to S. The S's TEE verifies the certificate and evaluates the

[4]Web API For Accessing Secure Element `https://globalplatform.github.io/WebApis-for-SE/doc/`

[5]Trustlite: `http://www.icri-sc.org/research/projects/trustlite/`

[6]ARMv8-M Architecture: `https://www.arm.com/products/processors/instruction-set-architectures/armv8-m-architecture.php`

compatibility of the T's TEE. If the verification succeeds, S's TEE encrypts the credentials using T's public key and transfers the encrypted credentials to T. However, the device certificate validation only guarantees the security and compatibility of the TEEs but does not offer user-to-device binding, i.e. it cannot ascertain that T and S belong to the same user.

Some credentials are confined to a particular device and cannot be migrated. For example, a SIM card stores a unique, non-migratable key for authenticating the SIM on a mobile network. Other credentials may incorporate policies, defined by their issuers, which limit sharing or migrating credentials according to the security needs. We can categorize migratable credentials into the following categories based on their transfer policies:

1. **Unrestricted credentials:** Some credentials do not incorporate any transfer restrictions or policies, such as, memorizable passwords and user-controlled PKI keys. Users can transfer these credentials between their devices or even share them with other users.
2. **User-bound credentials:** Some credentials are provisioned to devices only after ascertaining that the devices belong to a particular user, such as eIDs issued on users' mobile devices. These credentials can only be transferred to a compatible device owned by the same user.
3. **Single-instance credentials:** Some credentials can migrate from one device to another provided that only one copy of the credentials exist at any given time. For example, a public transport authority may allow users to migrate their ticketing credentials between devices. However, it cannot permit credentials to be simultaneously used from multiple devices to prevent double spending.

6.2.1 Transferring unrestricted credentials

We can use the straightforward approach to transfer TEE protected, unrestricted credentials between devices provided that the user/owner of S approve the transfer. However, this approach is not suitable for transferring other types of credentials as the approach lacks the user-to-device binding assurance.

6.2.2 Transferring user-bound credentials

Transferring user-bound credentials requires T to provide the credential issuer with a similar level of security assurance as S provided during initial provisioning. In other words, T must assure that, (a) it consists of a compliant TEE, and (b) prove that it belongs to the same user as S.

The first assurance is provided by device certificate validation (i.e. TEE compliance). For the latter we can use the user-to-device binding method of using existing hardware-based credentials (presented in Publication V, and Chapter 4). However, this method can only be used when a single issuer provisions all the credentials owned by a user. In such cases, the credential issuer only requires a single verification of user-to-device binding before transferring credentials. However, it results in a poor user experience when transferring credentials issued by multiple issuers. In other words, it requires each credential issuer to verify the user-to-device binding before transferring credentials.

For an optimal user experience, we need a design that requires verification of user-to-device binding only during the initial provisioning phase. Later, it should allow users to transfer credentials between devices only by proving that the same user owns T.

Kostiainen et al. [57] have proposed a user-friendly credential transfer protocol where they use a trusted server to store credentials as a backup. Users authenticate themselves to credential issuers only during initial provisioning. While transferring credentials, S copies unrestricted credentials to the trusted server which in turn restores them to T. However, the non-transferable credentials require re-provisioning on T from the corresponding credential issuers. For each non-transferable credential, S issues a delegated token to the trusted server. The trusted server issues further delegated tokens to T. Both S and the trusted server sign the delegated token. The signatures assure the credential issuers that T belongs to the same user as S. However, their solution uses passwords to provide user-to-device binding. Similarly, Ghada et al. [6] proposed a TEE migration protocol that relies on the user's consent in establishing a wireless channel between two devices to prove the ownership of the devices. However, the protocol does not provide assurance that T belongs to the same user.

These protocols are ideal for one-off migration of credentials from one device to another. However, they become unattractive for sharing and using credentials simultaneously on multiple devices. For example, a user must run the transfer protocols on all of her devices whenever she adds a new credential.

In contrast, OmniShare Publication X utilizes a user's cloud storage to create a *domain of authorized* devices for her and distributes as well as synchronizes credentials among these devices. The user can add a new device to the domain from any existing device in the domain. It utilizes a combination of an out-of-band channel (established with the user's consent) and the cloud storage service itself to authorize the new device for admission to the domain. We can utilize a similar approach to create a domain of trusted devices for an arbitrary user and allow sharing credentials between TEEs in the authorized devices owned by the same user. We can also use user-to-device binding with the user's hardware-based credential as mentioned in Publication V and Chapter 3. However, the question remains of how to design a globally acceptable user-to-device binding with users' hardware-based credentials. For example, as used in Publication V, FINeID can only be used to guarantee user-to-device bindings within the Finnish jurisdiction but may not be accepted as a valid mechanism for user identification outside of Finland.

6.2.3 Transferring single-instance credentials

To allow the transference of a single-instance credential between devices, we must assure that the TEE is compliant and also guarantee that the credential at S is inaccessible after the transfer. If the single-instance credential is also user-bound, then we must additionally ensure that T belongs to the same user.

Berger et al. [10] presented a virtual TPM (vTPM) migration technique where T generates a nonce to which S binds and locks its vTPM states. Once locked, the vTPM is inaccessible at S and can only be unlocked at T, if T can prove possession of the nonce. We can adapt a similar approach to the transfer of single-instance credentials and ensure that the credentials remain inaccessible at S after transfer. However, if the nonce at T gets corrupted before the completion of the transfer, or gets corrupted at S

before locking, the credentials cannot be recovered.

Marforio et al. [65] proposed a design that binds the credentials to users' phone numbers. They recommend modification to SIM applications to make them TEE-aware. During secure provisioning, the credential issuer sends an SMS containing the provisioning secret to a user's registered phone number. The SIM decrypts the provisioning secret and makes it available to the TEE. The secret then allows the TEE to initiate the provisioning protocol. The protocol requires the user to move her SIM to T for credential migration. Once the SIM is moved, the TEE on S deactivates the credentials. The credential issuer re-provisions the credentials to T in a similar way as the initial provisioning at S.

We can combine Marforio et al.'s approach with OmniShare to share single-instance credentials between all the devices in the domain, but gets activated only on a single device at any given time. In this approach, the TEE on a device detects whether the SIM is present and enables or disables the credentials on the device. Since users tie their credentials to a particular physical SIM, single-instance credentials can only be functional on a single device at a given time. However, this approach is limited to devices that can support a SIM. Overall, transferring single-instance credentials is still an open challenge.

6.2.4 Categorization of credentials discussed in this dissertation

- **eID credentials:** Publication I, Publication V and Chapter 3 presented credentials such as government-issued eIDs. eIDs are *user-bound credentials* that are used to authenticate users for accessing various online services. eIDs on devices are not strictly device-bound as users may change their device frequently. Publication V also shows the existence of eID credentials on multiple devices such as a physical smart card and a mobile device belonging to the same user. Therefore, users should be allowed to share their credentials among their devices.
- **Public transport credentials:** Credentials used in public transport ticketing as discussed in Publication II, Publication III, Publication IV and Chapter 4 are user-bound as well as single instance credentials. The credential issuers (i.e. the accounting authority)

may allow a user to transfer credentials from one device to another belonging to the same user (e.g. during device replacement). However, to avoid double spending, they can only allow the credentials to be usable from a single device at any given time.

6.3 Trusted Path

In some devices, both TEEs and REEs can access the user interface (UI). Therefore, it is important to protect security-sensitive information exchanged between a user and a TA. A secure communication interface between a user and a TA that protects the integrity and confidentiality of the information exchanged is called a *trusted path* [42]. A trusted path also assures the user that she is communicating with the TA and not with an impersonator [47].

6.3.1 Establishing a trusted path

To establish a trusted path between a user and a TA, the TEE must take explicit control over the device peripherals used in the trusted path. For example, the eID credentials in Publication I, Publication V, and Chapter 3 require a PIN before signing a document. The TEE should take pre-emptive control of the touchscreen while displaying the PIN-entry UI to protect the PIN against an adversary in the REE. Taking exclusive control over the touchscreen requires the trusted computing base (TCB) of the TEE to include the touchscreen driver.

In TrustZone, a separate secure OS kernel runs in the TEE. The secure OS can include device drivers needed for establishing trusted paths. However, adding support for multiple device drivers will bloat the TCB [99]. Unlike TrustZone, Intel's SGX does not run a separate secure OS in the TEE. Instead, SGX executes parts of applications that need to be protected as enclaves [67]. Furthermore, since the TCB for an enclave only includes the code running in the enclave and the CPU itself, SGX does not allow an enclave to make system calls [101]. Therefore adding a device driver support on each enclave becomes impractical. Intel does provide a proprietary trusted path called Protected Audio Video Path for securing high-definition video playback, which can only be used for displaying outputs securely, such as securing video conferencing outputs and displaying one-time passwords [43]. Weiser et al. [101] proposed an architecture called SGXIO that provides trusted path for application enclaves. The SGXIO uses a hypervisor to isolate IO device drivers from the REE. Further, SGXIO runs the part of device driver logic in a separate enclave, which allows other applications to share keys with the driver enclave using SGX's local attestation. An application enclave establishes a trusted path with the user via the device driver. The device driver enclave protects the messages exchanged over the trusted path by encrypting them with the shared key.

Alternatively, Sun et al. [91] suggested using a hypervisor to isolate the trusted and untrusted OS. The hypervisor protects the interaction between a user and the trusted OS from the untrusted OS. Their solution uses the power button as the "secure attention key", which activates the system BIOS for switching between trusted and untrusted OS. Similarly, Riedl et al. [82] suggested using gesture-based mechanisms to switch between security domains called *zones*. They presented three gesture-based switching mechanism (a) drawing patterns associated with particular zone on the touch screen of the device, (b) using device's lock screen with options to select zones, and (c) using multi-finger swipe on touch screen. They also conducted a user study that shows the lock screen-based switching as the preferred mechanism among the participants.

6.3.2 Authenticating a trusted path

It is also important to convey to users reliably that they are indeed using a trusted path to communicate with a TA. A graphical symbol (e.g. a logo of a padlock or shield) displayed on the address bar of a web browser is a prominent example of a *security indicator* (not for a trusted path) that indicates the security status of websites. However, many users are ignorant about the indicators or fail to notice them on web browsers [25].

Using similar graphical symbols as the security indicator for trusted paths on mobile devices can become even harder due to the smaller screen sizes compared to PCs. Moreover, the mobile web browsers hide the address bar to maximize the display of the web contents on the entire screen for optimal user experience. Moreover, fraudulent apps can imitate the UI used for displaying security indicators for phishing. Zhou et al. [105] reported that the majority of fraudulent apps are repackaged versions of legitimate apps with malicious code injected. Thus fraudulent apps can utilize UI components, such as login UI, PIN-entry UI, etc., from the original apps to spoof and relay the information collected through the UI to servers controlled by attackers.

6.3.3 Security indicators for TEEs

Various researchers have suggested using the status of LEDs on a device platform [62] or a color bar displayed at the top of the screen [85, 82], as the security indicator while interacting over a trusted path. Since the malicious application running on the REE of a device can spoof the security indicators displayed on the device's screen, Mayrhofer [66] emphasizes on using hardware-based security indicators that are directly controlled by TCBs.

The GlobalPlatform's Trusted User Interface API specification [35] attempts to standardize generic UI layouts for input/output of security-sensitive information on mobile devices. The specification suggests either using hardware controlled security indicators such as LEDs, or displaying information known only to the user and the TA, but not to the REE, as the security indicators.

Using dedicated hardware controlled by the TEE as the security indicator is expensive since it requires support from the device manufacturer and the replacement of current devices. Zhou et al. [106] proposed using TPM-based remote attestation to verify the status of a trusted path on a PC from an external device such as a mobile phone. Alternatively, users can share information such as a *secret passphrase* with the TEE safely during the boot process before the REE initiates [92]. Users are recommended to enter their security-sensitive information only on the UI that displays the shared passphrase established during the boot process.

6.3.4 Trusted path for applications discussed in this thesis

TAs that handle the eID credentials in Publication I, Publication II and Chapter 3 require a trusted path for entering a PIN. We can use the trusted PIN-entry UI, defined by GlobalPlatform [35], as the trusted path for PIN entry. We can further display a secret passphrase (shared between the user and the TEE) in the trusted PIN-entry UI as the security indicator. Recent TEEs offer trusted UI APIs to establish trusted paths for input/output operations [18]. However, the public transport ticketing protocols presented in Publication II, Publication III, Publication IV and Chapter 4 do not benefit from a trusted path between a user and the TA, as the TA does not need an endorsement from the user during identity verification. Also, reliably notifying users with a security indicator becomes challenging since the identity verification completes in a short interval, i.e. less than 300 milliseconds.

Privacy preserving computation on infrastructure such as the carousel approach discussed in Publication VI and also in Chapter 5 can utilize the trusted UI on users' devices to provide users with authentic information about their queries (e.g. whether or not an app is malicious, and how to remediate the potential threat). The trusted UI can also display a secret passphrase shared with the TEE on the infrastructure (over an out-of-band channel) as the security indicator. The trusted UI on the device protects the integrity of the results displayed against the manipulation from malware on the device.

6.4 Service augmentation with a combination of TEEs on devices and server infrastructure

As TEEs are available for both cloud infrastructure and users' devices, we can design services that utilize the TEEs on both sides to enhance security.

6.4.1 Device TEE as a front-end

A device's TEE can be used as a front-end for user input/output while performing actual computation on the cloud TEE. For example, a PMT protocol that allows users to check compromised service passwords can provide a TA to users' devices. The TA would allows users to enter their potentially compromised passwords via a trusted path and receive results securely. The TA lets users use the PMT service securely even in presence of malware on the device.

As another use case, we can use a device TEE as a front-end is to display one-time passwords. Users may require one-time passwords to access a corporate network. The corporation can host a password generation algorithm on its infrastructure TEE and distribute a client application to its users' devices. The client application only uses the TEE to display a trusted UI for displaying one-time passwords. This allows the corporation to maintain and update the password generation algorithm more efficiently without having to update applications on every client device.

TEE-aware SIMs, and vice-versa, as purposed by Marforio et al. [65] can also benefit from the standardized trusted UI. For example, security codes used in two-factor authentication are widely distributed via SMSs. While receiving a security code as an SMS, the TEE-aware SIM can notify the TEE to display the code via a trusted UI. Additionally, the UI can integrate a button for deleting the SMS securely after its use, or the SMS can be auto-removed safely after the security code expires. This would protect the two-factor authentication code distributed over SMS even in the presence of malware on users' devices. An important consideration while designing such services is that devices should have network capability to access such services.

6.4.2 Infrastructure TEE as credential backup

A TEE available in the cloud infrastructure can be used to build a credential backup service for users. The trusted server used in the credential migration protocol proposed by Kostiainen et al. [57] can be implemented as a TA running on the TEE of the service provider's cloud.

Further, the cloud TEE can be used to manage the domain of trusted devices in OmniShare Publication X. While adding a new device T to the domain, the TEE in T can interact with a device S in the domain over an out-of-band channel. Then both devices T and S can send their interaction evidence to the cloud TEE for the distribution of the root key to T without having to use the untrusted cloud storage as the communication medium. Also, while removing a device from the domain, the cloud TEE can generate a new root key and distribute it to all devices in the domain except the one that has been removed. The cloud TEE can also be used to assure deletion of single-instance credentials from S before they are transferred to T. The use of the cloud TEE also enhances privacy by preventing the (potentially untrusted) cloud provider from learning how many devices and what types of devices are present in a given user's domain.

6.5 Formal analysis of protocols

This dissertation uses standard protocols and cryptographic primitives whenever possible. However, many protocols and modified cryptographic schemes were introduced, especially in Chapter 4 (in Publication II, Publication III and Publication IV). Their correctness and security were only argued informally in the corresponding publications.

To provide stronger confidence, we need to analyze them using formal methods and automated analysis tools. This dissertation does not cover any formal verification in the interest of keeping the work tractable. This is left as future work. For example, **P4** uses the RSA signatures with message recovery to minimize the size of the certificate used as a part of the ticketing credential. This scheme should be verified using cryptographic proof techniques. Also, other protocols should be formally analyzed using automated tools such as ProVerif[7] or Tamarin Prover[8].

6.6 Conclusion

Trusted hardware is a critical security enabler that can be used to establish trust between various (non-trusting) entities in a multilateral environment. This dissertation showed how to use programmable trusted hardware, such as TEEs, to build protocols and applications for ensuring the security requirements of entities in multilateral scenarios. Specifically, it presented the use of a TEE at the *user's end* to fulfill the security requirements of service providers, and the use of a TEE within the *infrastructure* to ensure users' data privacy while accessing services on third-party infrastructure.

This dissertation used two use cases with real-world implications (i.e. hosting/using eID credentials on users' devices and using credentials for

[7]ProVerif: `http://prosecco.gforge.inria.fr/personal/bblanche/proverif/`
[8]Tamarin Prover: `http://www.infsec.ethz.ch/research/software/tamarin.html`

the public transport ticketing from NFC-enabled users' devices) to elaborate the use of TEEs at the user's end. Similarly, the use case of cloud-based malware checking set out an example of using a TEE to ensure users' data privacy while accessing services on untrusted infrastructure. The use cases were supported with significant efforts made in designing the systems, implementing them as prototypes on real TEEs (e.g. ARM TrustZone and SGX), and evaluating their performance.

In conclusion, the following substantial lessons from this dissertation can be summarized as: (a)Modern devices such as smartphones are better platforms for hosting credentials compared to physical smart cards in terms of security, cost, ease-of-management and usability. (b)With proper engineering effort we can design optimized yet secure identity-verification schemes that can be used over low bandwidth channels for applications such as public transport ticketing. (c)TEEs can be used to provide users' with privacy assurance while offering privacy-preserving services, such as a PMT, on untrusted servers.

Additionally, this dissertation presented other insights related to current TEE-based systems, which can be summarized as: (a) Beside a programmable trusted hardware, non-programmable trusted hardware can also be used as trust anchors to fulfill security requirements of several, but not all, applications. (b) Sharing/migrating credentials is a desirable feature. Some types of credentials are easy to share among other devices belonging to the same user while other types require stronger trust assurance. (c) A trusted path from a TEE would further improve security. At present, layouts and interfaces for trusted paths have been standardized. There should be more support from the TEE providers to allow developers to use trusted paths. (d) As TEEs are becoming available for various platforms, we can leverage a combination of TEEs on users' devices as well as the infrastructure to enhance the security of applications and further develop new types of services. This dissertation left the formal analysis of protocols as future work.

References

[1] Raja Naeem Akram, Konstantinos Markantonakis, and Keith Mayes. A Paradigm Shift in Smart Card Ownership Model. In *International Conference on Computational Science and Its Applications (ICCSA)*, pages 191–200. IEEE, 2010.

[2] Smart Card Alliance. Transit and Contactless Financial Payments: New Opportunities for Collaboration and Convergence. A Smart Card Alliance Transportation Council White Paper, October 2006. `http://www.smartcardalliance.org/resources/pdf/Transit_Financial_Linkages_WP_102006.pdf`. Accessed: 30-12-2016.

[3] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative Technology for CPU Based Attestation and Sealing. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, volume 13, 2013.

[4] Apple Inc. iOS Security – iOS 9.3 or later. iOS Security—White Paper, May 2016. `https://www.apple.com/business/docs/iOS_Security_Guide.pdf` Accessed: 31-01-2016.

[5] Claudio A Ardagna, Rasool Asal, Ernesto Damiani, and Quang Hieu Vu. From Security to Assurance in the Cloud: A Survey. *ACM Computing Surveys (CSUR)*, 48(1):2, 2015.

[6] Ghada Arfaoui, Saïd Gharout, Jean-François Lalande, and Jacques Traoré. Practical and Privacy-Preserving TEE Migration. In *International Conference on Information Security Theory and Practice*, pages 153–168. Springer, 2015.

[7] ARM. ARM Security Technology Building a Secure System using TrustZone® Technology. `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.prd29-genc-009492c/index.html`, 2009. Accessed 07-02-2017.

[8] Dirk Balfanz and Edward W. Felten. Hand-Held Computers Can Be Better Smart Cards. In *Proceedings of the 8th conference on USENIX Security Symposium*, pages 2–2, Berkeley, CA, USA, 1999. USENIX Association.

[9] Andrew Baumann, Marcus Peinado, and Galen Hunt. Shielding Applications from an Untrusted Cloud with Haven. *ACM Transactions on Computer Systems (TOCS)*, 33(3):8, 2015.

[10] Stefan Berger, Ramón Cáceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert Doorn. vTPM: Virtualizing the Trusted Platform Module. In *Proceedings of the 15th Conference on USENIX Security Symposium*, pages 305–320, Berkeley, CA, USA, 2006. USENIX Association.

[11] Burton H Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426, 1970.

[12] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct Anonymous Attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, CCS '04, pages 132–145, New York, NY, USA, 2004. ACM.

[13] Ernie Brickell and Jiangtao Li. Enhanced Privacy ID from Bilinear Pairing for Hardware Authentication and Attestation. *International Journal of Information Privacy, Security and Integrity 2*, 1(1):3–33, 2011.

[14] Victor Costan and Srinivas Devadas. Intel SGX Explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.

[15] Gerhard de Koning Gans, Jaap-Henk Hoepman, and Flavio D Garcia. A Practical Attack on the MIFARE Classic. In *International Conference on Smart Card Research and Advanced Applications*, pages 267–282. Springer, 2008.

[16] Damien Deville, Antoine Galland, Gilles Grimaud, and Sebastien Jean. Smart Card Operating Systems: Past, Present and Future. In *the 5th NORDU/USENIX conference*, volume 5, February 2003.

[17] EU Directive. 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. *Official Journal of the EC*, 23(6), 1995.

[18] Rob Dyke. The Benefits of Trusted User Interface. `https://www.trustonic.com/news-events/blog/benefits-trusted-user-interface`, September 2015. Accessed 07-02-2017.

[19] Jan-Erik Ekberg. Personal Communication. Trustonic, 2015.

[20] Jan-Erik Ekberg, N. Asokan, Kari Kostiainen, and Pasi Eronen. *OnBoard Credentials Platform Design and Implementation*, 2008.

[21] Jan-Erik Ekberg, Kari Kostiainen, and N Asokan. The Untapped Potential of Trusted Execution Environments on Mobile Devices. *IEEE Security & Privacy*, 12(4):29–37, 2014.

[22] EPC – GSMA. Trusted Service Manager Service Management Requirements and Specifications, January 2010. http://www.gsma.com/digitalcommerce/epc-gsma-trusted-service-manager-service-management-requirements-and-specifications-january-2010. Accessed: 07-02-2017.

[23] Ulfar Erlingsson, Mark Manasse, and Frank McSherry. A Cool and Practical Alternative to Traditional Hash Tables. In *Proceedings of the 7th Workshop on Distributed Data and Structures (WDAS'06)*, 2006.

[24] Eurostat. Internet and cloud services - statistics on the use by individuals, 2014. http://ec.europa.eu/eurostat/statistics-explained/index.php/Internet_and_cloud_services_-_statistics_on_the_use_by_individuals Accessed: 30-01-2017.

[25] Adrienne Porter Felt, Robert W Reeder, Alex Ainslie, Helen Harris, Max Walker, Christopher Thompson, Mustafa Embre Acer, Elisabeth Morant, and Sunny Consolvo. Rethinking Connection Security Indicators. In *Proceedings of the 12th Symposium on Usable Privacy and Security (SOUPS 2016)*, pages 1–14, Berkeley, CA, USA, 2016. USENIX Association.

[26] Lishoy Francis, Gerhard Hancke, Keith Mayes, and Konstantinos Markantonakis. Practical NFC Peer-to-Peer Relay Attack Using Mobile Phones. In *International Workshop on Radio Frequency Identification: Security and Privacy Issues*, pages 35–49. Springer, 2010.

[27] G Data. Mobile Malware Report, Threat report: Q4/2015. https://www.gdatasoftware.com/securitylabs/news/article/g-data-releases-mobile-malware-report-for-the-fourth-quarter-of-2015. Accessed: 30-11-2016.

[28] Flavio D Garcia, Gerhard de Koning Gans, Ruben Muijrers, Peter Van Rossum, Roel Verdult, Ronny Wichers Schreur, and Bart Jacobs. Dismantling MIFARE Classic. In *European Symposium on Research in Computer Security*, pages 97–114. Springer, 2008.

[29] Flavio D Garcia, Peter Van Rossum, Roel Verdult, and Ronny Wichers Schreur. Wirelessly pickpocketing a Mifare Classic card. In *IEEE Symposium on Security and Privacy*, pages 3–15. IEEE, 2009.

[30] Stefano Levialdi Ghìron, Serena Sposato, Carlo Maria Medaglia, and Alice Moroni. NFC Ticketing: A Prototype and Usability Test of an NFC-Based Virtual Ticketing Application. In *International Workshop on Near Field Communication*, pages 45–50. IEEE, 2009.

[31] GlobalPlatform. *Device Specifications for Trusted Execution Environment (TEE)*. http://www.globalplatform.org/specificationsdevice.asp. Accessed: 30-12-2016.

[32] GlobalPlatform. TEE Client API Specification Version 1.0. Device Specifications, July 2010. http://www.globalplatform.org/specificationsdevice.asp. Accessed: 30-12-2016.

[33] GlobalPlatform. TEE System Architecture version 1.0. Device Specifications, December 2011. http://www.globalplatform.org/specificationsdevice.asp. Accessed: 30-12-2016.

[34] GlobalPlatform. A New Model: The Consumer-Centric Model and How It Applies to the Mobile Ecosystem. White Papers, March 2012. http://www.globalplatform.org/documents/Consumer_Centric_Model_White_PaperMar2012.pdf.

[35] GlobalPlatform. Trusted User Interface API. Device Specifications, June 2013. http://www.globalplatform.org/specificationsdevice.asp. Accessed: 30-12-2016.

[36] GlobalPlatform. *GlobalPlatform Card Specification v2.3*, December 2015. http://www.globalplatform.org/specificationscard.asp. Accessed: 30-12-2016.

[37] GlobalPlatform. TEE Internal Core API Specification Version 1.1.1. Device Specifications, June 2016. http://www.globalplatform.org/specificationsdevice.asp. Accessed: 30-12-2016.

[38] GlobalPlatform. TEE Management Framework Version 1.0. Device Specifications, November 2016. http://www.globalplatform.org/specificationsdevice.asp. Accessed: 30-12-2016.

[39] Oded Goldreich and Rafail Ostrovsky. Software Protection and Simulation on Oblivious RAMs. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[40] Google. Android Keystore System. https://developer.android.com/training/articles/keystore.html.

[41] Google. Android Security 2015 Year In Review, April 2015. http://static.googleusercontent.com/media/source.android.com/en//security/reports/Google_Android_Security_2015_Report_Final.pdf.

[42] David Grawrock. *Dynamics of a Trusted Platform: A Building Block Approach*. Intel Press, 1st edition, 2009.

[43] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. Using Innovative Instructions to Create Trustworthy Software Solutions. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, page 11, 2013.

[44] Intel. Software Guard Extensions SDK: Developer Reference, 2016. https://software.intel.com/en-us/sgx-sdk/documentation. Accessed: 15-02-2016.

[45] ISO. ISO/IEC 14443 Identification cards – Contactless integrated circuit cards – Proximity cards (Part 1 - 4).

[46] ISO. ISO/IEC 15693 Identification cards – Contactless integrated circuit cards – Vicinity cards (Part 1 - 3).

[47] ISO. *ISO/IEC 15408-1:2009: Information technology – Security techniques – Evaluation criteria for IT security – Part 1: Introduction and general model*. International Organization for Standardization, Geneva, Switzerland, 2009.

[48] ISO. *ISO/IEC 21481:2012: Information technology — Telecommunications and information exchange between systems — Near Field Communication Interface and Protocol (NFCIP-2)*. International Organization for Standardization, Geneva, Switzerland, 2012.

[49] ISO. *ISO/IEC 18092:2013: Information technology — Telecommunications and information exchange between systems — Near Field Communication — Interface and Protocol (NFCIP-1)*. International Organization for Standardization, Geneva, Switzerland, 2013.

[50] ISO. *ISO/IEC 7816-4:2013: Identification cards — Integrated circuit cards — Part 4: Organization, security and commands for interchange*. International Organization for Standardization, Geneva, Switzerland, 2013.

[51] ITSO. Interoperable public transport ticketing using contactless smart customer media. Version V2.1.4. ITSO Technical Specification 1000, May 2016. `https://www.itso.org.uk/the-specification/specification-resources/publicly-available-specification/`. Accessed: 07-02-2017.

[52] JEDEC Solid State Technology Association, EMBEDDED. Embedded MultiMediaCard(e.MMC). JEDEC Standard.

[53] JIS. *JIS x 6319-4 Specification of implementation for integrated circuit(s) cards — Part4: High Speed proximity cards*. Japanese Industrial Standard, Tokyo, Japan, 2010.

[54] J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447, RFC Editor, February 2003.

[55] Alexey Kirichenko. Personal Communication. F-Secure, 2015.

[56] Kari Kostiainen. *On-board Credentials: An Open Credential Platform for Mobile Devices*. PhD thesis, Aalto University, May 2012.

[57] Kari Kostiainen, N. Asokan, and Alexandra Afanasyeva. Towards User-Friendly Credential Transfer on Open Credential Platforms. In *International conference on Applied cryptography and network security*, pages 395–412. Springer, 2011.

[58] Kari Kostiainen, N. Asokan, and Jan-Erik Ekberg. Practical Property-Based Attestation on Mobile Devices. In *International Conference on Trust and Trustworthy Computing*, pages 78–92. Springer, 2011.

[59] Kari Kostiainen, Jan-Erik Ekberg, N Asokan, and Aarne Rantala. On-board Credentials with Open Provisioning. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, pages 104–115. ACM, 2009.

[60] 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 1st ACM conference on Data and application security and privacy*, pages 13–24. ACM, 2011.

[61] Pekka Laitinen. *HTML5 and Digital Signatures: Signature Creation Service 1.0.1*, June 2015. `https://eevertti.vrk.fi/fineid-maaritykset`. Accessed: 30-12-2016.

[62] Wenhao Li, Mingyang Ma, Jinchen Han, Yubin Xia, Binyu Zang, Cheng-Kang Chu, and Tieyan Li. Building Trusted Path on Untrusted Device Drivers for Mobile Devices. In *Proceedings of 5th Asia-Pacific Workshop on Systems*, page 8. ACM, 2014.

[63] Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard. ARMageddon: Cache attacks on mobile devices. In *Proceedings of the 25th USENIX Security Symposium*, pages 549–564, 2016.

[64] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-Level Cache Side-Channel Attacks are Practical. In *2015 IEEE Symposium on Security and Privacy*, pages 605–622. IEEE, 2015.

[65] Claudio Marforio, Nikolaos Karapanos, Claudio Soriente, Kari Kostiainen, and Srdjan Capkun. Secure Enrollment and Practical Migration for Mobile Trusted Execution Environments. In *Proceedings of the 3rd ACM workshop on Security and privacy in smartphones & mobile devices*, pages 93–98. ACM, 2013.

[66] René Mayrhofer. An architecture for secure mobile devices. *Security and Communication Networks*, 8(10):1958–1970, 2015.

[67] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative Instructions and Software Model for Isolated Execution. In *International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, pages 10:1–10:1, New York, NY, USA, 2013. ACM.

[68] Tommi Meskanen, Jian Liu, Sara Ramezanian, and Valtteri Niemi. Private Membership Test for Bloom Filters. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 515–522. IEEE, 2015.

[69] Srijith K Nair, Andrew S Tanenbaum, Gabriela Gheorghe, and Bruno Crispo. Enforcing DRM Policies Across Applications. In *Proceedings of the 8th ACM workshop on Digital rights management*, pages 87–94. ACM, 2008.

[70] Thomas Nyman, Jan-Erik Ekberg, and N. Asokan. Citizen Electronic Identities using TPM 2.0. In *Proceedings of the 4th International Workshop on Trustworthy Embedded Devices*, pages 37–48. ACM, 2014.

[71] Thomas Nyman, Sandeep Tamrakar, Lari Lehtomäki, Jere Vaara, Pawel Sarbinowski, Sami Jaktholm, and N. Asokan. On Deploying TEE-based Authentication. Poster presented at the Secure Systems Annual Demo

Day, Aalto University, June 2015. https://wiki.aalto.fi/display/sesy/2015+Secure+Systems+Annual+Demo+Day.

[72] Oracle. Java Card Technology. http://www.oracle.com/technetwork/java/embedded/javacard/overview/index.html. Accessed: 30-12-2016.

[73] Rasmus Pagh and Flemming Friche Rodler. Cuckoo Hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

[74] PC / SC Workgroup Specifications. Interoperability Specification for ICCs and Personal Computer Systems - Part 1-10. https://www.pcscworkgroup.com/specifications/. Accessed 31-1-2017.

[75] Gábor Pék, Levente Buttyán, and Boldizsár Bencsáth. A Survey of Security Issues in Hardware Virtualization. *ACM Computing Surveys (CSUR)*, 45(3):40, 2013.

[76] Lau Peter Shiu Cheung. Developing a Contactless Bankcard Fare Engine for Transport for London. Master's thesis, Massachusetts Institute of Technology. Dept. of Civil and Environmental Engineering, 2009.

[77] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private Set Intersection Using Permutation-based Hashing. In *Proceedings of the 24th USENIX Security Symposium*, pages 515–530, Washington, D.C., 2015. USENIX Association.

[78] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster Private Set Intersection Based on OT Extension. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, pages 797–812, Berkeley, CA, USA, 2014. USENIX Association.

[79] Himanshu Raj, Stefan Saroiu, Alec Wolman, Ronald Aigner, Jeremiah Cox, Paul England, Chris Fenner, Kinshuman Kinshumann, Jork Loeser, Dennis Mattoon, et al. fTPM: A Firmware-based TPM 2.0 Implementation. *Microsoft Research Technical Report 2015-84*, November 2015.

[80] Wolfgang Rankl and Wolfgang Effing. *Smart Card Handbook*. John Wiley & Sons, 2004.

[81] Kai Rannenberg. Multilateral security a concept and examples for balanced security. In *Proceedings of the 2000 Workshop on New security paradigms*, pages 151–162. ACM, 2001.

[82] Peter Riedl, Rene Mayrhofer, Andreas Möller, Matthias Kranz, Florian Lettner, Clemens Holzmann, and Marion Koelle. Only play in your comfort zone: interaction methods for improving security awareness on mobile devices. *Personal and Ubiquitous Computing*, 19(5-6):941–954, 2015.

[83] Ahmad-Reza Sadeghi, Marko Wolf, Christian Stüble, N Asokan, and Jan-Erik Ekberg. Enabling Fairer Digital Rights Management with Trusted Computing. In *International Conference on Information Security*, pages 53–70. Springer, 2007.

[84] Nuno Santos, Krishna P Gummadi, and Rodrigo Rodrigues. Towards Trusted Cloud Computing. *HotCloud*, 9(9):3, 2009.

[85] Marcel Selhorst, Christian Stüble, Florian Feldmann, and Utz Gnaida. Towards a Trusted Mobile Desktop. In *International conference on Trust and trustworthy computing*, pages 78–94. Springer, 2010.

[86] Suranga Seneviratne, Aruna Seneviratne, Prasant Mohapatra, and Anirban Mahanti. Predicting User Traits from a Snapshot of Apps Installed on a Smartphone. *ACM SIGMOBILE Mobile Computing and Communications Review*, 18(2):1–8, 2014.

[87] William Shapiro and Radek Vingralek. How to Manage Persistent State in DRM Systems. In *ACM Workshop on Digital Rights Management*, pages 176–191. Springer, 2001.

[88] Smart Card Alliance. Transit and Contactless Open Payments: An Emerging Approach for Fare Collection. A Smart Card Alliance Transportation Council White Paper, November 2011. http://www.smartcardalliance.org/publications-transit-financial-2011/ Accessed: 30-11-2016.

[89] Jay Srage and Jérôme Azema. M-shield mobile security technology. TI White paper, 2005.

[90] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an Extremely Simple Oblivious RAM Protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 299–310. ACM, 2013.

[91] Kun Sun, Jiang Wang, Fengwei Zhang, and Angelos Stavrou. SecureSwitch: BIOS-Assisted Isolation and Switch between Trusted and Untrusted Commodity OSes. In *the 19th Annual Network & Distributed System Security Symposium*, 2012.

[92] Tianhao Tong and David Evans. Guardroid: A Trusted Path for Password Entry. *Proceedings of Mobile Security Technologies (MoST)*, 2013.

[93] Hien Thi Thu Truong, Eemil Lagerspetz, Petteri Nurmi, Adam J Oliner, Sasu Tarkoma, N Asokan, and Sourav Bhattacharya. The Company You Keep: Mobile Malware Infection Rates and Inexpensive Risk Indicators. In *Proceedings of the 23rd international conference on World wide web*, pages 39–50. ACM, 2014.

[94] Trusted Computing Group. TCG Infrastructure Workgroup Subject Key Attestation Evidence Extension, Specification Version 1.0, Revision 7. TCG Published, June 2005. https://www.trustedcomputinggroup.org/wp-content/uploads/IWG_SKAE_Extension_1-00.pdf Accessed: 30-12-2016.

[95] Trusted Computing Group. TPM Main Specification Level 2, part 1-3, Version 1.2, Revision 116. TCG Published, March 2011. http://www.trustedcomputinggroup.org/tpm-main-specification/. Accessed: 30-12-2016.

[96] Trusted Computing Group. Trusted Platform Module Library Specification, part 1-4, Family "2.0", Level 00, Revision 01.16. TCG Published, October 2014. http://www.trustedcomputinggroup.org/tpm-library-specification. Accessed: 30-12-2016.

[97] Trustonic. Kinibi Trusted Execution Environment (TEE). https://www.trustonic.com/products/kinibi. Accessed: 30-11-2016.

[98] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic rowhammer attacks on mobile platforms. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1675–1689. ACM, 2016.

[99] Amit Vasudevan, Emmanuel Owusu, Zongwei Zhou, James Newsome, and Jonathan M McCune. Trustworthy Execution on Mobile Devices: What security properties can my mobile platform give me? In *International Conference on Trust and Trustworthy Computing*, pages 159–178. Springer, 2012.

[100] Population Register Centre (VRK). FINEID S4-1 Implementation profile 1 for Finnish Electronic ID Card, v2.1a. FINEID specifications, October 2004. https://eevertti.vrk.fi/fineid-maaritykset. Accessed: 30-12-2016.

[101] S. Weiser and M. Werner. SGXIO: Generic Trusted I/O Path for Intel SGX. *ArXiv e-prints*, January 2017.

[102] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In *2015 IEEE Symposium on Security and Privacy*, pages 640–656. IEEE, 2015.

[103] Yahoo Aviate. How Android users interact with their phones. https://yahooaviate.tumblr.com/image/95795838933, August 2014. Accessed: 30-11-2016.

[104] Ka-Ping Yee. User Interaction Design for Secure Systems. In *International Conference on Information and Communications Security*, pages 278–290. Springer, 2002.

[105] Yajin Zhou and Xuxian Jiang. Dissecting Android Malware: Characterization and Evolution. In *2012 IEEE Symposium on Security and Privacy*, pages 95–109. IEEE, 2012.

[106] Zongwei Zhou, Virgil D Gligor, James Newsome, and Jonathan M McCune. Building Verifiable Trusted Path on Commodity x86 Computers. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 616–630. IEEE, 2012.

Aalto-DD 105/2017

ISBN 978-952-60-7464-1 (printed)
ISBN 978-952-60-7463-4 (pdf)
ISSN-L 1799-4934
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)

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