5-12-2024

# A TrustZone-based Framework to Secure Mobile Financial Transactions and Provide End-to-End Protection for QR-code Payments and Credit Card Information

Ammar Salman Salman
*Syracuse University*

# Abstract

In this work we have developed multiple solutions for financial transactions that can be coordinated to provide high level of security and data integrity, while providing all services with minimum changes in infrastructures, and maximum flexibility. The solutions are novel in many aspects and the generalization is a clear feature. The TrustZone hardware is an important component to ensure high security and protection. The system can function smoothly on any type of operating systems and works with any platform of services in the market. The tested case study is built on the Android system and the ARM TrustZone hardware. This particularly does not mean it is the best option. Under other operating systems the design can work equally efficiently or better.

Types of transactions can include using peripheral devices like cameras, GPS, or NFS that can work in connection with mobiles or standard systems. QR code payment are fully served including merchant or buyer modes with static or dynamics payments. Credit card options are also fully served. The offline generation of virtual credit card systems plays a major role in providing uniqueness, high security, and maximum flexibility to serve all platforms.

In the development and testing, we have fully experimented with four component solutions. The first covers using peripheral devices, the second provides decoding capacity within the TrustZone and eliminating the need for decoding by external servers. the Virtual Credit Card (VCC) offline dynamic generation of numbers provides maximum flexibility in offering secure payment and eliminating middleman services. Finally, the addition of encoding in the TrustZone generates secure QR codes covering the buyer presented option. The VCC offline dynamic generation one way hash

encryption and decoding ensures to get secure information. Only the user and the bank parties can reconstruct and confirm the true authenticity and accuracy of the communication. Moreover, the use of UTC time to narrow the window of validity of the credit card number closes known gaps for the conduct of unauthorized transactions.

With such conjunction, we developed a framework design to build a scalable commercial and industrial applications that can use the current infrastructure, while getting the best of all services. An important requirement is to have a cooperating TrustZone hardware provider and a banking system willing to implement the system. The framework can be applied smoothly to serve different operating systems and payment services.

To get an alternative, the framework can be served even more efficiently and securely if we construct a TrustZone compatible standalone TrustProvider SOC. Such system can remove the requirement of having a side to cooperate with. In fact, it provides more flexibility, because there is no need to work with certain manufacturers. The system is in design and development as an ongoing work. Such system is still TrustZone based, and all components are managed by a Trusted Execution Environment (TEE). Meanwhile, it can also provide the TrustZone service using the mobile phones albeit with less security in terms of communication.

**Keywords:** ARM TrustZone, TrustProvider, DOT-VCC, Mobile Security, REE, OPT-TEE, QR payments, OP-TEE, Android, Threat Model, Attack Surface. ZBar Decoder, Split QR decoding, Payment fraud prevention. VCC generators, Encoding systems. Buyer Present QR code.

# A TrustZone-based Framework to Secure Mobile Financial Transactions and Provide End-to-End Protection for QR-code Payments and Credit Card Information

I/O Access and Data Security/Integrity; Split SSL Decoding and Encoding
DOT-Virtual Credit Card Generation with Security/Integrity.
A Framework with TrustZone Compatible SoC TrustProvider.

by

Ammar S. M. Salman

B.E., Alquds University, 2016

M.S., Syracuse University, 2021

Dissertation

Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy in Electrical and Computer Engineering.

Syracuse University

June 2024

# Acknowledgements

I am pleased to thank my supervisor, Professor Kevin Du for his support and guidance. Thanks extend to the department of Electrical Engineering and Computer Science at Syracuse University and the cybersecurity group.

I am also indebted to thank the reviewing committee, Professor Riyad S. Aboutaha, Professor Yuzhe Tang, Professor Garrett Katz, and Professor Mostafa Taha for showing constant support and understanding and providing valuable feedback. Also, I would like to give special thanks to Dr. Kailiang Ying and Dr. Amit Ahlawat for offering help whenever was needed.

I extend my special thanks to my father Professor Salman M. Salman who as always provided guidance, encouragement, and deep insights for me. Special thanks to my mother Professor Suhair H Salman for her love and daily support. Sister Abeer lives in Jordan, but I always feel we are together. I am specially indebted to brother Odai (who completed PhD in deep learning in medical applications) for the cooperation and support during our years of undergraduate and graduate studies and the hospitality when we exchange visits. Finally, special thanks to sister Adan who has completed her graduate studies in computer engineering from Syracuse university, and to Obada my youngest brother who is studying computer science at SUNY POLY. I look forward to collaborating with all of them in the future.

Ammar Salman

Syracuse, NY 2024

Table of Contents

## List of Figures

## List of tables

# List of Abbreviations and Terms Definitions

| Term | Meaning |
|---|---|
| Android | The mobile operating system. |
| API | Application Programming Interface. |
| ARM | Advanced RISC Machines: a family of CPUs based on the RISC architecture developed by ARM. Comes in 32-bit and 64-bit multi-core processors. |
| Attack Surfaces | The total number of threat that the system is exposed to and their success frequency. |
| CVV | Card Verification Value |
| Decoding | Extracting embedded information from representation. E.g., QR decoding extracts the original text that was represented through encoding as a QR image. |
| Decoded data | Tools that control the form of data from comprehensible or not. |
| DoS | Denial of Service. |
| DOT-VCC | Dynamic Offline TrustZone Virtual Credit Card |
| Emulator | A hardware device or software program that enables a host computer system to imitate the functions of another Guest computer system. |
| FW | Firewall. |
| GPS | Global Positioning System. |
| HTTP | Hypertext Transfer Protocol. |
| I/O devices | Input/output devices including camera GPS NFC…etc. |
| LoC | Lines of Code. |
| Location attestation | Identifying the location through GPS or other means. |
| Luhn algorithm | A formula used to calculate checksums and it's commonly used to calculate the last of the 16 digits in credit cards |
| NFC | Near Field Communication. |
| NIST | National Institute for Standards and Technology |

| | |
|---|---|
| NVD | National Vulnerability Database |
| One way hash | A process where information is transformed irreversibly into another representation where only holders of the original information may generate the same representation. |
| OS | Operating system. |
| Performance overhead | Efforts in terms of time and resources that are added to the normal operation due to certain changes in the original settings and the effect on the standard operations |
| QR Code | Quick Response code |
| REE | Rich Execution Environment. |
| RISC | Reduced Instruction Set Computer. |
| SEEM-FTF | Secure End-to-End Mobile Financial Transactions Framework |
| SoC | System on Chip |
| Split-SSL-handshake | Secure Sockets Layer. Instead of relaying an insecure HTTP connection, an SSL splitting proxy simulates a normal SSL connection with the client by merging authentication records from the server with data records from a cache. |
| TA | Trusted application. |
| TAC | Trusted application client (aka CA) |
| CA | Client Application (aka TAC) |
| TCB | Trusted Computing Base. |
| TEE | Trusted Execution Environment. |
| Threat Model | The main weak spots in the security system that allow certain breaches of the normal operations of the original system. |
| Tokenization | The process of passing tokens that represent information without passing the information directly. |
| TrustZone | An operating system that ensures hardware and software full protection. |
| TruZ-Droid | A design and implementation using TrustZone to offer secure services for UI and SSL/ |

| | |
|---|---|
| TrustProvider | An SOC design for a standalone TrustZone based device that supports secure I/O |
| UART | Universal Asynchronous Receiver-Transmitter. |
| UI | User Interface. |
| Verification | Certain protocol of test to verify if the implementation works as intended. |
| Vulnerabilities | Weak spots in the system that allow breaches. |
| VCC | Virtual Credit Card |
| ZBar | Open-source C library for barcode and QR code decoding |

# Chapter 1

# Introduction, Problem Statement, and Contributions

# Chapter 1: Introduction, Problem Statement, and Contributions

## 1.1 Introduction

The security problem for real-time applications is a challenge that must take into account the variability of the attack types and frequency [1]. Operating systems constantly provide security updates, but they lag far behind attackers who exploit these systems, which leads to vulnerabilities affecting sensitive operations users perform. One focus in this work is on security procedures for Android devices utilizing TrustZone. The role of the TrustZone is to provide dedicated protection that can foil attacks by preventing accessibility to critical information or operations. The TrustZone concept provides a successful utility for different operating systems environments [2], [3], [4], [16]. This work focuses on providing security for financial transactions conducted using mobile devices using TrustZone.

In this section, we first define the problem statement, and the needed protection for the Android user intensive applications, such as payment, location, and integrity of data. We introduce the ARM TrustZone, and the specialized TruZ-Droid developed by our group [2], [5], [6].

The work of this thesis covers four components of financial or general transactions that use various methods like QR coding encoding and virtual credit card generation and decoding. A framework out of these components is discussed. All components are tested to operate in the TrustZone, or an alternative standalone TrustProvider SoC, which is proposed to be dedicated for these functions under the supervision of a dedicated hardware trust environment. Using a standalone TrustProvider brings more dedication and independence. Still, the operations are compatible with regular TrustZone.

Section 1.2 states the overall problem statement and general objectives, and the details will be thoroughly covered when discussing each component definition, design, and implementation in

the following chapters. The thesis is divided into seven chapters as listed in section 1.3. Chapters 1 and 2 are introductory and background, and Chapters 3-6 cover the core components description, ending with a framework proposal that integrates the components under a single trust management. Finally, Chapter 7 presents the overall conclusions and proposals for future work or extensions. Section 1.4 summarizes the contributions and publications.

### 1.1.1 Motivation

1. Mobile devices are increasingly used for financial transactions, and credit card payments are vulnerable to remote attacks causing sensitive data loss.
2. Major financial losses result in global credit & debit card fraud losses amounting to $34B in 2022 [1], [11], [12] [75].
3. Hackers need only 6 seconds to breach the secrets as practiced in the existing payment systems, security codes and expiry dates are easily bypassed [11].
4. Merchant and service provider databases are vulnerable to attacks, and credit card theft and manipulation.
5. Complex infrastructure offers more room for attack surfaces and vulnerabilities.
6. Tokenization, virtual credit cards, service-provider-based transactions need an overall revision using Arm TrustZone based protections.
7. This work addresses countering these problems on mobiles systems using built-in or standalone TrustZone devices.

### 1.1.2 ARM TrustZone

1. A secure enclave created by ARM implemented software and hardware.
2. The TrustZone offers two execution environments: The secure Trusted Execution Environment (TEE) and the unsecure Rich Execution Environment (REE).

3. The hardware-based isolation ensures data stored in the TEE is inaccessible to the REE, which makes it suitable for storing information like biometric data and payment information.
4. The TEE is flashed on the device from the factory and does not receive Over-The-Air updates, making the TEE inaccessible as a service to many applications that may want to utilize its security. The TEE is kept small to ensure it only supports core features.
5. App developers must communicate with the manufacturers to have support for the features they desire. The process is expensive, and most application developers cannot afford it.
6. Developing generic services that can be utilized by application developers seamlessly is a goal.
7. TrustZone systems are available extensively on mobile REE systems, but can be utilized independently.

### 1.1.3 Existing Credit Card Protection Techniques

**Tokenization**

1. User-end tokenization: user passes token to the merchant who validates the correctness of the token and uses service-providers in the credit card network to fulfill the transaction. Service-providers map the token to the real credit card being used so that merchants do not need to hold the credit card info.
2. Merchant-based tokenization: the merchant gets the card information from the user and the merchant is responsible for tokenizing the data using service-providers who store the credit card information.
3. Tokenization is heavily dependent on service-providers (middlemen) to function.

**Virtual Credit Cards (VCCs)**

1. VCCs are useful in masking real credit card data with virtual data that can be set to expire after a transaction, rendering theft of that data near harmless, because the original data cannot be extracted from it.
2. VCCs are exclusively used for online payments and are still vulnerable to remote attacks against devices used by service-based transaction providers.

**Service-based transaction providers**

1. Users often rely on service-providers to offer a platform for financial transactions.
2. They offload their security to that platform, which stores their credit card information and uses it to generate platform-specific transactions in the form of QR codes that can be scanned by other users for payment processing (buyer-presented or merchant-presented QR codes).
3. Such platforms can be managed by the banks or operating on their own, and they do support VCCs for online transactions.
4. The primary issue is that the platforms are increasingly becoming targets for remote attackers, given that they contain valuable data. Breaches against them do lead to theft of many users' credit card data.

### 1.1.4 Summary of Our Framework

1. With several protection techniques introducing middlemen to mitigate merchant-related breaches, they also introduce new risks against their own databases. How do we mitigate the risks for users without changing the infrastructure?
2. We introduce a framework that eliminates the need of service-providers' security, but incorporates the developed Dynamic Offline Trusted Virtual Credit Card generation (DOT-VCC) for personal and online payments and maintains the integrity/secrecy of the data through the ARM TrustZone protection.

3. This offers end-to-end transactions that do not expose any real credit card information in the infrastructure aside from the issuing bank.
4. The system eliminates data theft through middleman-related breaches. The breaches can still happen, but no useful data is stolen.
5. The DOT-VCC works with existing credit card network infrastructure without modification to the core operations.
6. The framework offers seamless integration for developers, transparent user experience, and protects integrity and secrecy of data based on the need.

## 1.2 Problem Statement & Objectives

### 1.2.1 Problem and Solution

**Problem**: How to protect user information in financial transactions, both in person and online, from remote attacks against mobile devices as well as merchant and service provider databases, while maintaining a transparent user experience, seamless integration for developers, secrecy of sensitive data, and integrity of transactions?

**Solution**: Building secure services for the TrustZone to cover different aspects of the financial transactions and integrate them in the TrustZone without introducing a large code base that makes the TrustZone vulnerable to attacks. Functions include encoding/decoding, peripheral control, and dynamic offline generation of virtual credit cards (DOT-VCC) for secure use. The framework should facilitate end-to-end transaction security that does not expose users' data.

**Constraints:** Transparent user experience, as added security should not inconvenience user experience. Another concern is seamless integration, where developers employing security measures should not have to redesign everything in their development pipelines. Finally,

solutions should not add large amounts of code into the TEE, which is the idea of a Small Trusted-Computing-Base (TCB).

### 1.2.2 Objectives

Our main objective is building a framework for TrustZone based integrated transaction systems for mobile devices, which also offers a foundation for a TrustZone compatible standalone SoC TrustProvider.

Specific underlying objectives

1. Securing the data path for merchant-presented transactions to protect data integrity.
2. Processing the transaction data in the TrustZone while maintaining small TCB.
3. Maintaining the secrecy of user data in buyer-presented transactions.
4. Providing secure presentation of buyer-presented transactions for processing.
5. Offering end-to-end transaction security that does not rely on the security of service providers.

### 1.2.3 Outcomes

1. Assured data integrity of camera and location devices.
2. Split-QR Decoder: hybrid design for TrustZone QR decoding operation.
3. DOT-VCC: TrustZone-based VCC generator for financial transactions.
4. QR-Encoder: TrustZone-based QR encoder for DOT-VCC data.
5. Framework: integrated framework combining the outlined components offering secure financial transactions, using the TrustZone, and can be built a standalone (TrustProvider).

### 1.2.4 Threat Model Basic Assumptions

1. The security of the REE is fully compromised.
2. Ordinary communication channels are not secure.

3. Compromised merchant databases or other service-provider systems, with impacts limiting existing infrastructure-based countermeasures, like tokenization.
4. The TrustZone provides high security over data residing in the TEE and can completely shut down the REE's access to resources it needs.

### 1.2.5 Threat Model Attack Surfaces

1. The mobile device: REE consists of millions of lines of code and loaded with features that make it vulnerable to remote attacks by gaining full root access over the device, including memory buffers, and I/O, resulting in stealing any data that flows through the OS. This threat model is explored in depth in Chapters 3 and 4.
2. Provider/merchant databases: such databases are attractive for attackers. Companies that host them are often victims of ransom, and in other cases the data is sold on the dark web. This threat model is explored in further depth Chapters 5 and 6.

A solution against both threats necessitates a framework encompassing the whole transaction paths. Our solution eliminates all threat model vulnerabilities.

## 1.3 Manuscript Structure and Organization

The thesis is divided into seven chapters. This introductory chapter identifies objectives, main assumptions, and accomplished outcome. Chapter 2 covers background about TrustZone and our main objectives and application that are used in the development of the main products. The background includes decoding, encoding concepts, TruZ-Droid [2] (contains Split-SSL and TruZ-UI) used in communication and decoding/encoding functions. Chapter 3 covers the first component, namely the development of a secure I/O under TEE operation. It defines the threat model, the solution design, the implementation test and evaluation of the actual feasibility of the product on industrial scale. Chapter 4 covers the second component, which is the development

Split-QR Decoder hybrid design for QR decoding in the TrustZone. Table 1.1 shows the list of chapters and titles.

Table 1.1 List of chapters and titles.

| Chapter | Title |
|---|---|
| Chapter 1 | Introduction and problem statement |
| Chapter 2 | General background. |
| Chapter 3 | Assured data Integrity of camera and location devices [7] |
| Chapter 4 | Split-QR Decoder: hybrid design for ARM TrustZone [8] |
| Chapter 5 | VCC generation with security and integrity (DOT-VCC) [9] |
| Chapter 6 | Secure End-to-End Mobile Financial Transactions Framework (SEEM-FTF) [10] |
| Chapter 7 | Conclusions and Future Work. |
| Citations | References. |

Chapter 5 presents the third component which is the novel design DOT-VCC for credit card protection. Chapter 6 covers the fourth component of encoding in relation to QR codes and DOT-VCC generation with a proposed design for a framework covering the four components and managed under the regular TrustZone on mobiles. It also discusses a design for standalone TrustProvider that works under TEE independent of manufacturers settings. Finally, Chapter 7 outlines the conclusions, recommendations, and future work.

## 1.4 Contribution: A Brief Summary of the Thesis Work

**Solutions and Outcomes**

Outcomes 1-2 of section 1.2.3 target the user-end vulnerabilities, while 3-5 target both user-end and service-related vulnerabilities. Outcome 5 also offers integration enabling end-to-end secure transactions. Contributions are outlined as follows:

### 1.4.1 Secure camera QR Payment Capture and Transfer to Servers for Processing (Ch.3).

1. Split-camera configuration capability that lets the REE configure the camera and focus settings, while the TEE takes the capture. This ensures the integrity of the capture is never compromised.
2. Secure data path from the peripheral all the way to recipient server. This component delegates the processing of the QR code to the server by sending the image capture to the server. This eliminates any processing on the TrustZone side but introduces overhead.

### 1.4.2 QR Split Decoder that Works Within the TrustZone. (Ch.4)

1. Split-QR decoder design that allows the TrustZone to directly process QR codes without needing to delegate the task to remote servers offering improved performance.
2. The design processes meta-data in the REE but leaves the extraction of the data to the TEE and operates only on the original copy of the data which maintains the integrity of the retrieved data.

### 1.4.3 DOT-VCC Generator & SEEM-FTF Framework (Ch. 5 & Ch. 6)

1. Unique design that combines the benefits of VCCs and tokenization without relying on service providers for the security of financial transactions which include tokenization and VCC service providers.
2. The design offers capability for end-to-end transactions without users exposing any sensitive information about their financial data to others. The circle of trust demanded by this design only contains the bank issuing the card, and the user who has the card.
3. It works for both physical transactions as well as virtual online transactions given the DOT-VCC is suitable for both types.

4. The design is still compatible with existing infrastructure and does not require any modifications to merchants nor service-provider systems to work with them. The design just renders their stored data irrelevant in cases of security breaches.
5. It protects against compromised user devices by providing the service in the TrustZone. The design also makes the service accessible to any banking system that wishes to utilize it without adding application-specific code in the TEE, thereby maintaining the security of the TEE while adding major security benefits.
6. The design introduces QR encoding within the TrustZone that is very lightweight and does not introduce any significant performance overhead. The encoder ensures the added security does not inconvenience user experience.
7. The Secure End-to-End Mobile Financial Transactions Framework (SEEM-FTF) offers utilization of encoding and decoding capabilities in connection with DOT-VCC to create secure peer to peer activities.

### 1.4.4 Published Work. List of published articles covering this work

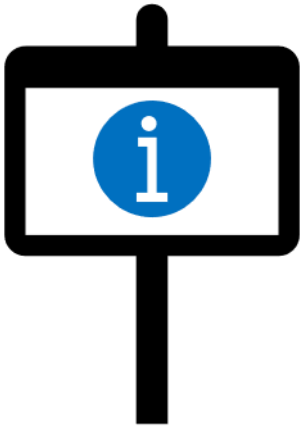
1. Salman, A.S., Du, W. (2021). Securing Mobile Systems GPS and Camera Functions Using TrustZone Framework. In: Arai, K. (Ed.) Intelligent Computing. Lecture Notes in Networks and Systems, vol 285. Springer, Cham. https://doi.org/10.1007/978-3-030-80129-8_58
2. Salman, A.S., Du, W.K. (2022). Split-QR Decoder Hybrid Design for ARM TrustZone. In: Arai, K. (Ed.) Advances in Information and Communication. FICC 2022. Lecture Notes in Networks and Systems, vol 439. Springer, Cham. https://doi.org/10.1007/978-3-030-98015-3_64
3. Salman, A.S., Du, W.K. (2022). Dynamic Offline TrustZone Virtual Credit Card Generator for Financial Transactions. In: Arai, K. (Ed.) Advances in Information and Communication.

FICC 2022. Lecture Notes in Networks and Systems, vol 439. Springer, Cham. https://doi.org/10.1007/978-3-030-98015-3_65

4. Salman, A.S., Du, W.K. (2024). A Framework for TrustZone Encoding/Decoding for QR Buyer-Presented and VCC Offline Generated Transactions, submitted to Future Technologies Conference 2024.

# Chapter 2

# General Background



# Chapter 2: General Background

## 2.1 Introduction

Modern operating systems are built on a massive legacy infrastructure that consists of millions of lines of code (LoCs) accumulating over decades of development. The primary focus throughout the development of such code bases is feature additions and fixing bugs, with a secondary concern for security-related analysis. While modern OSes are much better equipped to deal with attacks, due to constant support and security patches, they are still considered quite vulnerable against cybersecurity threats. Security analysis is expensive and can hardly cover all cases in such massive systems. Therefore, security patches are always playing catch-up with hackers discovering new bugs and vulnerabilities. This makes such systems very unreliable for security-critical operations, such as biometric authentication, financial transactions, and handling sensitive data.

Luckily, the cybersecurity threat was recognized by many developers and researchers, and the concept of a secure enclave, that is only used for security-sensitive operations, was born. Different developers have different implementations of such an enclave, and they are used for different use-cases. However, all such implementations are primarily hardware-based to eliminate a lot of threats against the enclaves themselves.

ARM introduced the ARM TrustZone in all its processors, and the TrustZone has been available in nearly every Android mobile device on the market for the past decade. However, the utility of the TrustZone is very limited and does not cover many security-sensitive operations. Expansion of that utility is also a difficult problem because the expansion itself can cause threats against the TrustZone itself, which if breached, nullifies all protections for users and their data.

The threats against operating systems affect security for all sorts of transactions, including financial transactions that use QR codes for convenience of the users. Figure 2.1 shows the general overview of the types of QR payments. Chapter 3 explains, in more detail, how the QR transactions take place.

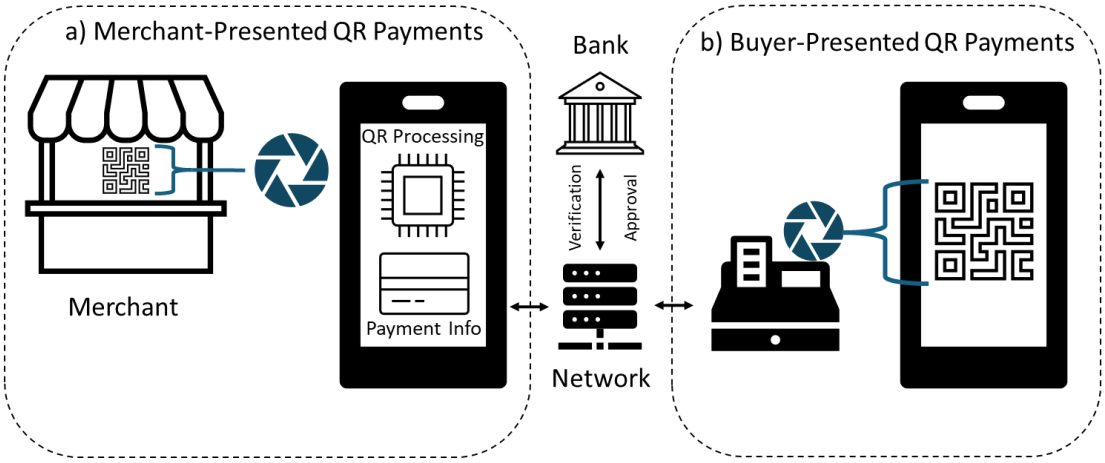



Figure 2.1: QR code payment types.

In this chapter, we introduce background about Android-related attacks, the ARM TrustZone, our research group work utilizing the TrustZone, and QR payment systems.

Section 2.2 covers the general threat model, and section 2.3 reviews the ARM TrustZone in relation to our work. Section 2.4 reviews our group developed a modified version of a TrustZone named TruZ-Droid. Section 2.5 outlines core encoding functions with a follow up in Chapter 6. Section 2.6 lists decoding with a detailed review in Chapter 4, and sections 2.7 and 2.8 list tokenization and DOT-VCC offline credit generation, which are covered in Chapter 5.

## 2.2 General Threat Model

The general threat model the thesis addresses includes two major attack areas, namely, attacks against user mobile devices, and attacks against service-related databases that store users' critical information.

Mobile devices are generally assumed unsafe for data security and sensitive data protection due to the wide range of attacks that exist against them. Attacks do not need physical access to the device to break through its security systems, and gain access to its content. Figure 2.2 shows the threat model

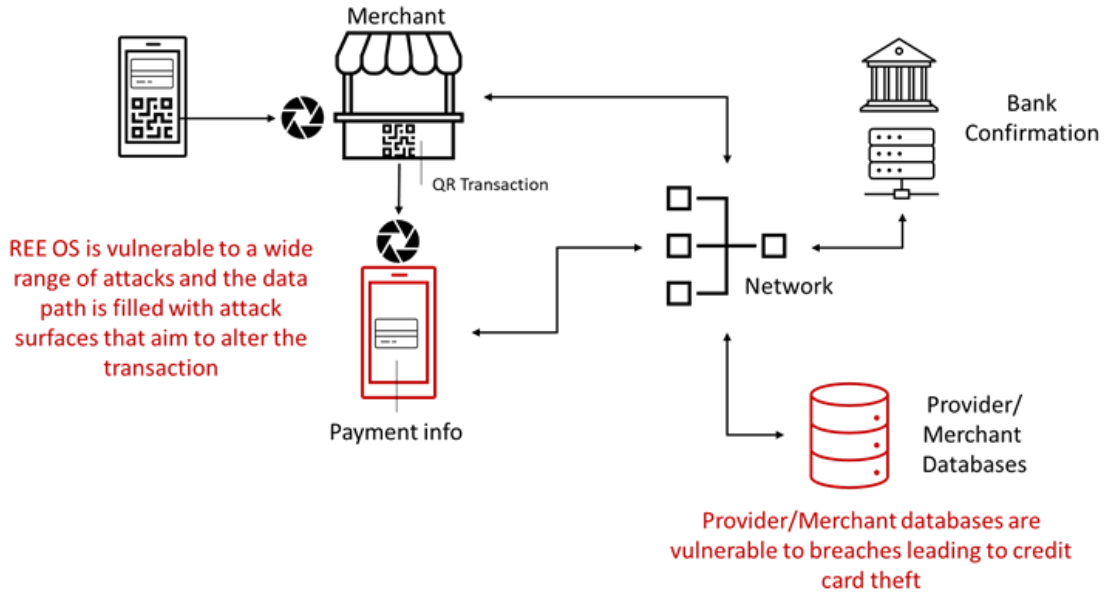



Figure 2.2: General Threat Model

Any security related application needs to address the shortcomings of mobile security through other means. On the other side, when users utilize services that involve financial transactions, those services are vulnerable to remote attackers who like to steal data from the databases and use it for malicious purposes. That includes financial information like credit cards. The thesis work introduces protections against both attack categories and aims to protect users' financial data when operating under compromised mobile devices, and with vulnerable service-related databases.

### 2.2.1 Android Breaches and Corruption

There is a large number of vulnerabilities in Android OS. The trend in discovery of new vulnerabilities is slowing down, but the number is already very high. Attacks and vulnerabilities

cover a large range of possibilities that range from attacking small components, to attacks against the infrastructure of the OS that yield root access to remote attackers. Such access gives attackers full control over every data path throughout the OS, and can lead to massive breaches of sensitive data. Despite the constant security patches, Android OS is still unreliable for sensitive data operations. Such security breaches are not limited to Android OS, they also apply to iOS where there are several examples of breaches that gain attackers full control over the device.

Figure 2.3 shows the yearly number of vulnerabilities related to Android OS according to the National Institute of Standards and Technology's (NIST) National Vulnerability Database (NVD) [1],[13]. Whenever using Android OS, we need to also be mindful of vulnerabilities that are not disclosed to the NVD [14], given that the procedure to disclose is typically done by ethical hackers who discover and report the vulnerabilities instead of exploiting them. Malicious actors do not operate in the same manner. Therefore, it is reasonable to assume that any

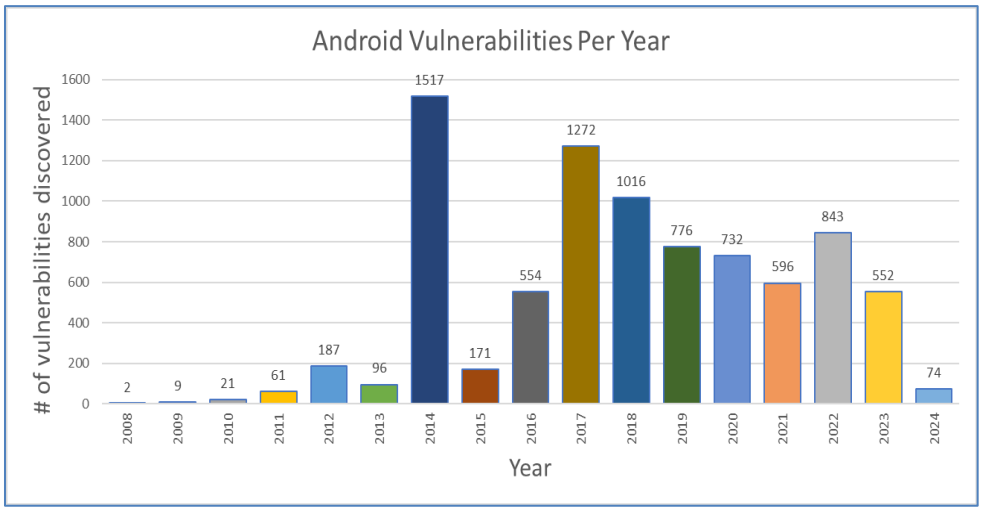



Figure 2.3: NIST's NVD database - Android vulnerabilities per year [1]

operation conducted under Android is vulnerable, and that includes financial transactions.

Accordingly, mobile devices utilize secure enclaves for sensitive operations. Android uses ARM's TrustZone while iOS uses Apple's Secure Enclave. Given a wider range of companies

develop Android-based devices, the specific implementation of the TrustZone, both in software and hardware, differs from one manufacturer to the other [2], [3].

The secure enclave offers better security countermeasures that can eliminate several risks in principle. For example, they are the industry standard for processing fingerprint data as the sensor is hardwired to the enclave which blocks the normal OS from accessing fingerprint information. Other applications include cryptography, wallets, payments, and more.

All commercial Android devices, in the market today, are equipped with ARM's TrustZone. The TrustZone has its own operating system, developed by the vendors. Such systems do not have updates given they are imprinted into secure isolated hardware components during the manufacturing process. Only the vendor can update such systems, and that requires physical access to the devices. Vendors do not disclose their specific implementations of the TrustZone operating systems for added security.

### 2.2.2 Credit/Debit Card Theft

Credit and debit card fraud and attacks have been very costly to the financial markets around the world. It was estimated that the credit/debit card fraud in 2022 reached 33 billion dollars in losses [12]. The credit card infrastructure is massive and financial transactions have many endpoints which are vulnerable to attacks and breaches that leak users' financial data. A survey conducted by W. Ahmed et. al. found it only takes 6 seconds to gain access to the credit card information in online transactions [11]. The attacks are not limited to online transactions because through everyday use, the users' financial data is stored in several places on the credit card network. Merchants who accept payments may store the credit card information in their databases, which then become targets for attackers. Other merchants delegate their responsibility for security by using service-providers to process the transactions for them, and accordingly they

do not store any credit card information in their databases. However, the problem is just moved and not solved because service-providers become the target for attackers, and breaches against them are even more damaging given they contain more credit card information than merchants.

While there are several security mechanisms put in place to protect users credit card information, the problem persists, and it is increasingly getting worse as users rely on credit cards more than cash [12]. Countermeasures include tokenization which comes in two variants: user-end and merchant-based tokenization schemes [69], [70], [71]. Tokenization relies on service-providers for translation of tokens into real credit card numbers, but as explained earlier, this does not offer protection against service-provider breaches. There are also virtual credit cards which offer better degree of protection due to their nature in hiding credit card information altogether by using virtually generated ones [62], [63], [64]. VCCs are used in online transactions and are not suited for physical use due to the security limitations of mobile devices.

## 2.3 ARM TrustZone

The ARM TrustZone [3], [4], [16] is a hardware-based security enclave, based in the processor, which offers security through hardware design that splits the execution environment into two: Trusted Execution Environment (TEE) and Rich Execution Environment (REE). The TrustZone design splits the physical core in a processor into two virtual cores, one is considered a secure core while the other is a normal core. There is a Secure Monitor which is responsible for the switching between the environments through a Secure Monitor Call (SMC) which suspends the context and clears all the resources for the switch. The secure world has its own set of resources which are never accessible by the REE, those resources are where the operating system of the TEE resides and operates. Such resources include certain regions of the main memory and cache. Furthermore, the TEE can also access any resource in the entire device on demand. The

monitor's responsibility is to completely block the REE from accessing any resource being accessed by the TEE.

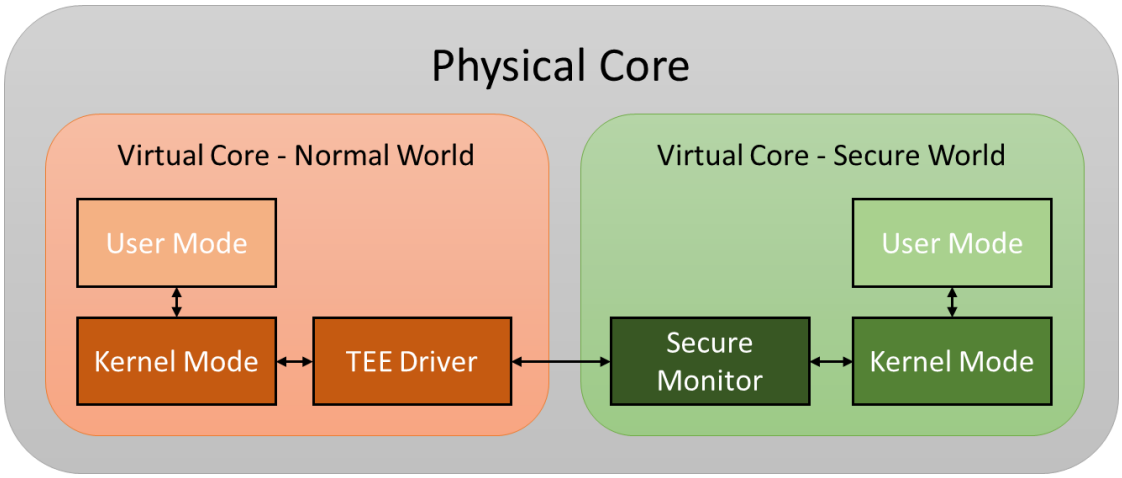



Figure 2.4: Physical core layout under ARM TrustZone design.

For example, if the TEE needs to operate on a REE's main memory for a specific function, the REE is not allowed to read/write from that region until the TEE relinquishes control back to the REE. It is the responsibility of the TEE and the secure monitor to ensure any data that remains there may not be used to deduce any secrets contained in the TEE. The TrustZone can be implemented in several ways as the implementation stretches to the entire System-on-Chip (SoC) design which includes the processor, cache, memory, data lines, storage and peripherals. The SoC designers generally follow the guidelines by ARM for their specific implementation and the context switching between the two worlds (secure and normal) needs to be in all such implementations. Figure 2.4 shows how the physical core is split and the general layout of the two execution environments (or worlds). The typical REE-TEE implementation requires TEE to boot and initialize first, then REE is follows.

The boot sequence is shown in Figure 2.5. The TEE contains a Trusted-Computing-Base (TCB) which relies on small size to maintain security analysis feasible by including only necessary functionality [4]. Maintaining a small TCB is critical to maintain security of the TEE. And in

general, developers avoid adding legacy code into the TCB because such code is hard to verify for security applications. There are several operating systems for the TEE including Google's Trusty OS [15] and GlobalPlatform's Open-source Portable Trusted Execution Environment (OP-TEE) [15]. Trusty OS uses more modern languages for the source-code, including Rust, while OP-TEE relies solely on C99 standard.

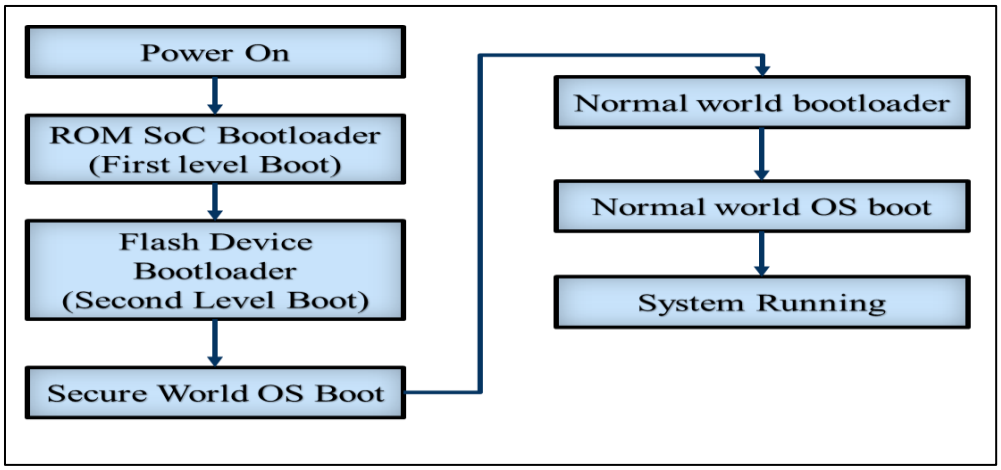



Figure 2.5: Boot sequence for the TEE and REE [2]

The TEE systems offer their own APIs and dispatchers which allow the REE to make use of their services, which include encryption, wallets, biometric authentication, and others. An application that has specific security needs will have to use those APIs to protect the sensitive operations.

The TEE and REE have their own structures which look similar in terms of both having kernel and user spaces for operations. The kernel is dedicated to the OS services which include management of devices and drivers, management of TCP/IP and so on.

Applications run in the user space. When the TEE provides a service, it is implemented as a Trusted Application (TA) which operates within the user space in the TEE and relies on the TEE's kernel to function. On the other side in the REE, there has to be a client application, typically referred to as the Trusted Application Client (TAC) or Client Application (CA) which runs in the user space of the REE and is responsible for sending requests to the TA, which are sent through the dispatcher (or daemon) that manages the TEE API calls in the REE.

While Trusty OS and OP-TEE are open-source, vendors (e.g., Samsung, Sony... etc.) have their own specific implementations of such systems, and they are closed-source and proprietary. This limitation means any third-party developers who would utilize the TEE services are limited in the capacity provided by the vendors and the TEEs. Typically, third-party application developers have to communicate with the vendors to have certain parts of their operations embedded in the TEE to offer them security. This process is very expensive and lengthy, and is not feasible for the majority of developers who do not have large budgets, nor the capacity to communicate with vendors to have their code reviewed and added to devices during manufacture. Furthermore, when such code is added, it does not necessarily support other applications who wish to use the same security features, which adds even more complexity. It is very desirable to have services be generic and accessible to everyone while offering security, and the theme of this thesis is to follow that principle. The TrustZone is adopted for use in a wide range of applications including internet [17], [18] general communication and networks [19] zero trust interactions [20] [21] and use in Internet of Things (IoT) [22].

## 2.4 TruZ-Droid

Kailiang Ying, et. al. [2], [31] introduced the TruZ-Droid design which expands the utility of the TEE to include several services including Split-SSL and TruZ-UI. The design focused on making the services as generic as possible which makes them support any application that wishes to utilize the TEE for security sensitive operations. The services do not contain any application specific code in the TEE, and they allow for easy integration for application developers who wish to use them. The design was integrated into the Android OS's SDK and APIs which allows applications to use those services like they would any other service.

TruZ-Droid allows applications to benefit from the TEE to protect the user's input confirmation, and sending the confidential information to the authorized server. They have built a prototype using the TrustZone-enabled Hi-Key board to evaluate the design. TruZ-Droid evaluation showed applications can leverage TrustZone while using existing OS APIs. They claim users can safely interact with TruZ-Droid to protect sensitive activities and data.

### 2.4.1 Split-SSL

The Split-SSL [2], [31], [32], [33] design allows for the secure transfer of data between the TEE and remote servers without REE manipulation of the data, and it does so without introducing large TCB into the TEE. When applications send data to servers, they use HTTPS to pack the data and send it over the network. HTTPS is built upon TCP and the SSL protocols. The SSL handshake is critical for secure communication with remote servers. However, the challenge is using those protocols within the TEE under a very limited TCB. The TCP/IP protocol implementation in the Linux kernel, which Android is built upon, contains hundreds of thousands of LoCs, the SSL protocol adds even more LoCs which make porting their implementation infeasible to achieve in the TEE. The Split-SSL design recognized the limitation and aimed to split the HTTPS communication between the REE and the TEE. They highlighted the key functions necessary for the secure transport of data and only ported them to the TEE while leaving the rest of the large code base in the REE.

Figure 2.6 shows the SSL handshake and the data encryption process in the Split-SSL design. The first step is the verification of the certificate received from the remote server.

This step is necessary to protect against Man-in-the-Middle (MITM) attacks, and it has to be done within the TEE in order to establish a root of trust for the handshake. In other words, the TEE verifies it is talking to the correct server through the X509 verification step. It then returns a

Boolean which tells the application in the REE whether the verification was successful or not. Then the normal handshake operation begins with the application requesting Pre-Master-Secret (PMS) from the TEE. Then the TEE uses a random number generator to create the PMS then it sends a reference back to the REE. The reference does not contain the PMS, it is just a pointer that the TEE can recognize internally. The SSL process continues for the Master Secret (MS) and the session keys (private key, shared secret, and session key) are generated and agreed upon without exposing any of their information to the REE, thereby ensuring the REE cannot act as the MITM attacker throughout this handshake, which further ensures the REE cannot decrypt the encrypted traffic between the TEE and the server.

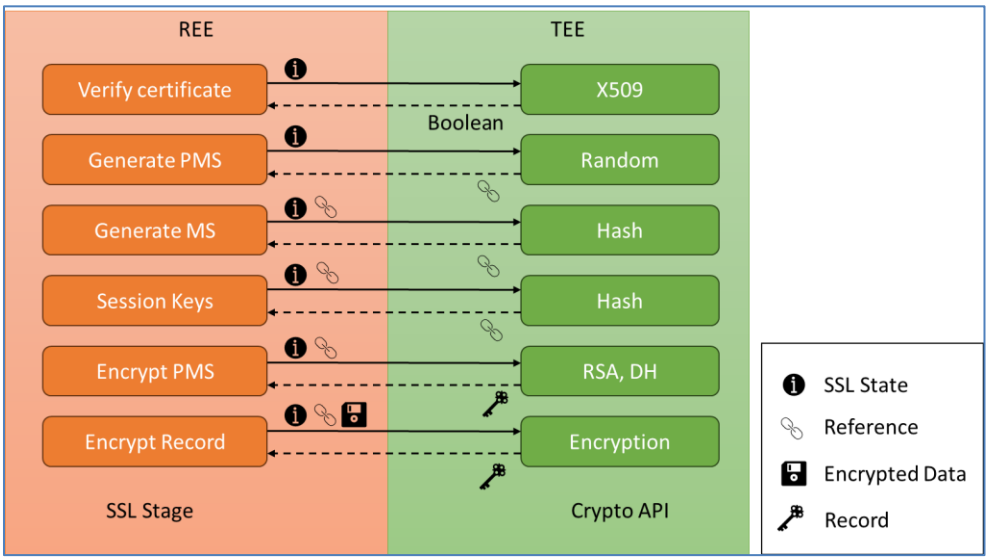



Figure 2.6: Split-SSL design - SSL Handshake & Encryption [2]

The Split-SSL design also addresses the TLS data size limits, and it breaks down large packets to smaller ones that can be individually sent to the server. The Split-SSL design also employs HTTP header attestation to allow the remote server the ability to verify that the REE is not engaging in a MITM attack before sending data to the TEE where it is eventually decrypted. This means the server can also trust the communication with the TEE and be aware that the data coming from the TEE is legitimate. The server's awareness of the TEE's attestation is a critical

step because it helps servers to reject data that was processed through the REE in cases where no TEE was used (for example, in Denial-of-Service attacks that intend on getting services through the REE instead of the TEE). However, this can lead to service unavailability, but the Split-SSL design did not consider availability against DoS attacks as part of the design's scope.

### 2.4.2 TruZ-UI

The TruZ-Droid extensions also include the TruZ-UI which allows for a transparent use experience while operating within the TEE. The design focuses on providing Android UI within the TEE but without actually using any of the Android Framework code base to do so.

Figure 2.7 shows their UI. The design recognizes that the TEE is only used for security sensitive operations like displaying secrets or obtaining secret data from the user [31], [32], [33].

Therefore, it does not need to offer a UI rendering capability all within the UI to achieve that goal. Instead, the design relies on using screenshots of the REE's display, then it modifies the screenshot to include the secret information after the REE has been locked out of accessing the display device. This is done by directly accessing the framebuffer in the main memory, which is a resource typically given to the REE.

When the TEE needs access to the display, it seizes control over the framebuffer, and it alters the data contained within it to display the secret data. In their case, they used this design to display a barcode and a QR code that were already stored within the TEE for the demonstration. Obtaining the QR and barcode was not the focus of the work, nor the demonstration.

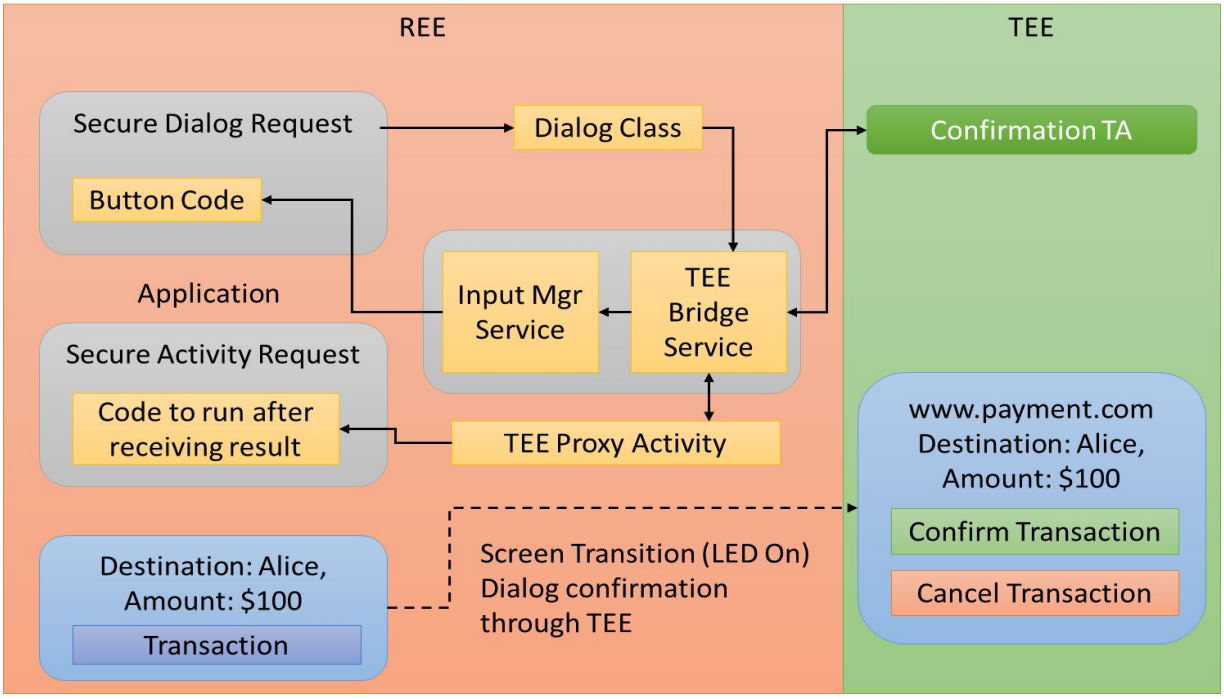



Figure 2.7: Confirmation UI Integration of the TruZ-Droid [2].

Furthermore, the TruZ-UI design also allows users to interact with the TEE through the keyboard TA that they have built for the TEE as shown in figure 2.8. The keyboard. TA also makes use of the screenshots concept where the CA (client application) renders the screen that wishes to take user's input, then requests the service from the TEE.

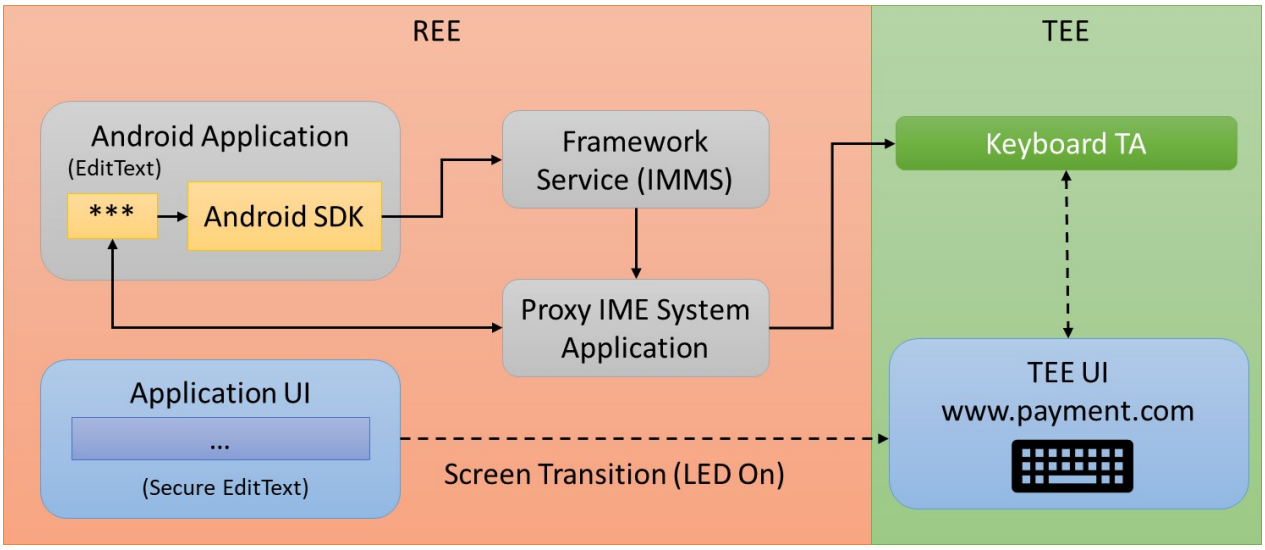



Figure 2.8: Keyboard integration of the TruZ-Droid [2]

The frozen framebuffer will also include the keyboard as rendered by the REE, but the TEE will have its own overlay that allows it to receive input from the touchscreen to properly record the data being fed by the user. This means we can rely on this service to securely add data to the TEE without worrying about the REE obtaining the data, and while maintaining transparent user experience throughout the process.

## 2.5 QR Code Encoding Functions

In this section we review a simpler encoding with minimum derivation details. For derivations please refer to [23], [24], [25], [26], [27], [28], [29]. Chapter 6 covers the implementation we have used with additional references and discussion.

### 2.5.1 Definition

Quick Response (QR) refers to a two-dimensional bar code bit matrix created by the Japanese company Denso-Wave. They are generally scanned by mobile and other devices, with applications that use the device camera to scan the code. The data stored in a QR Code can contain product information and price, direct to links, provide contact information, or compose a text-message or email [23], [24], [26].

As figure 2.9 shows, QR Codes are square grids filled with black and white pixels. Three of the corners have distinctive concentric squares, called Finder Patterns. The pixels squares that make up the symbol are called modules. Each module represents one bit of data, white or black, meaning 0 or 1, respectively. It's possible to use colors provided there is sufficient contrast.

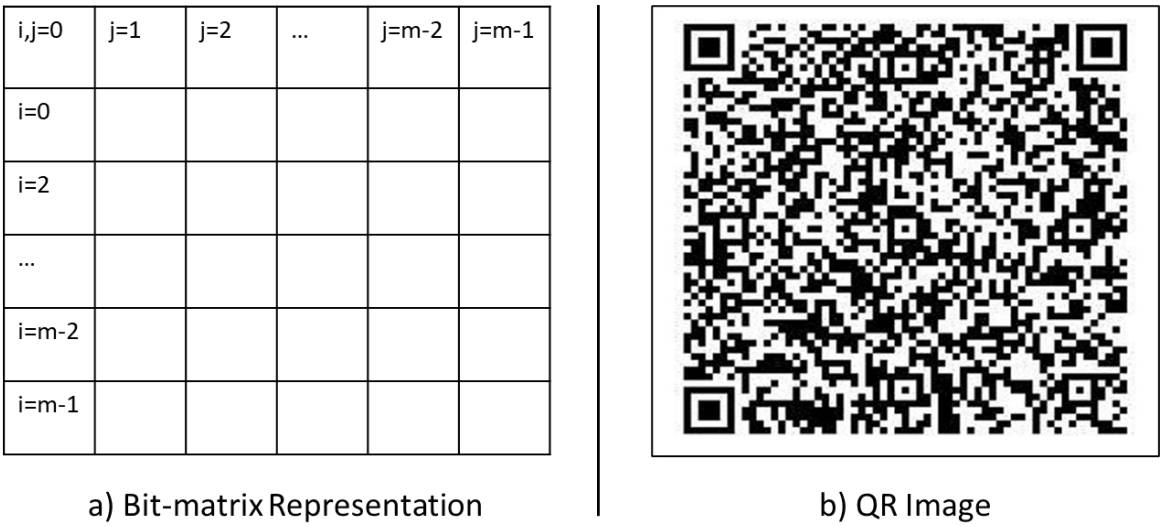



Figure 2.9: Bit matrix representation and a real QR image [23]

There are 40 QR Code versions, which refer to the size. Version 1 is 21x21 modules, and each successive Version increases by 4 modules in each dimension, all the way up to Version 40, which is 177x177 modules. Each version is surrounded by blank white space with thickness of 4 modules. There's no limit on the size of the images, they can be a fraction of a millimeter, or several feet square. But the practical size is around 5x5 $cm^2$.

The image is made of m x m matrix, where m=21+ 4n, n=0…39; min m = 21, max m = 177. i= rows and j= columns; element = $e_{ij}$ = 1bit. Figure 2.9 shows the representation and a QR image.

All 40 versions support 4 types of Error Correction (EC), namely L, M, Q, and H. L supports roughly 7% of data to be restored, M (the default) supports restoration of about 15%, Q supports about 25%, and H, 30%. Higher EC modes allow more data recovery in case the symbol is damaged, but require using more modules for EC.

### 2.5.2 QR Code Structure

Figure 2.10 shows the different sections of a QR symbol. In what follows we discuss the stored information in the various sections [24].

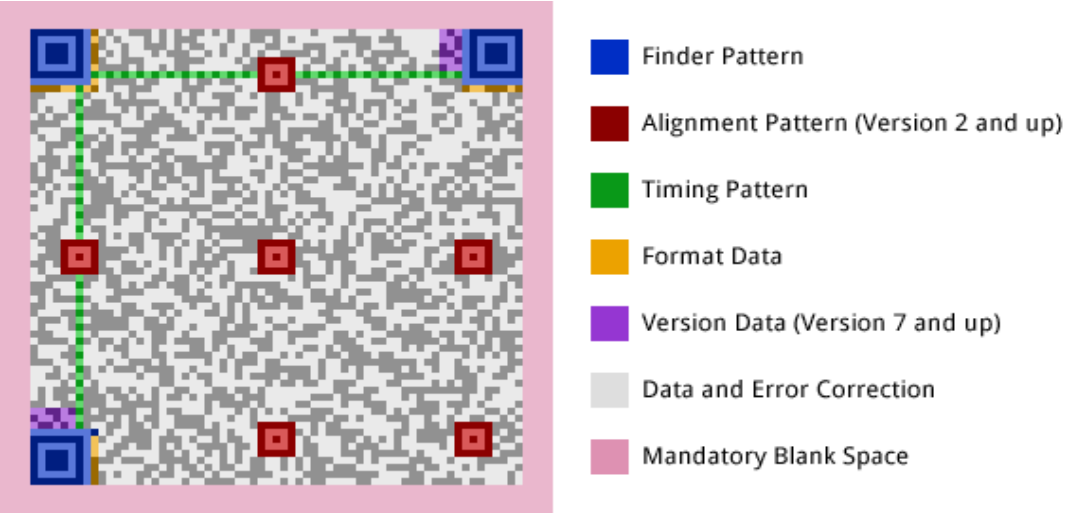



Figure 2.10: QR code structure [23], [24]

**Alignment Pattern**: for versions 2 and higher, alignment patterns are placed to help decoders adjust skewing in the symbol. Without them, it would be harder for a decoder to convert a skewed photograph into the virtual grid of data. The alignment patterns are also concentric squares like the finder patterns, but the center is a single black module. It's surrounded by a one-thick white box, which is surrounded by a one-thick black box, with no white space outside. Higher versions have more alignment patterns placed across the symbol.

**Timing Pattern**: an alternating stripe of black and white modules is located vertically and horizontally between the finder patterns. Starting with black on the innermost black corner of the finder patterns, the Timing Patterns alternate values toward the adjacent finder patterns. There are two lines to the timing pattern - one running horizontally between the two top finder patterns, and one running vertically between the two left-side finder patterns (the green lines in Figure 2.10).

**Format Data:** the data in the QR symbol is masked by reversing rows and columns. In order for the decoder to know which modules to read as reversed, the selected mask is stored twice, along with the error correction version. After the format information is encoded to 15 bits, it's placed in

the symbol twice. The first copy starts on the far left, directly beneath the white space around the top-left finder pattern (row i = 8, column j = 0, zero-indexing). The first (highest-order) bit is placed at this position, and subsequent bits are placed to the right. The module of the timing pattern in the way is skipped over, two more bits are placed to the right, then the data turns around the corner of the finder pattern's white space.

The modules from the other timing pattern are also skipped, and the remaining modules are placed vertically until the 15th bit (the lowest-order) is placed at the top row i =0 of the symbol. The second copy starts from the bottom just to the right of the white space around the bottom-left finder pattern. (Last row i = m-1, column j = 8, zero-indexed) The bits are placed from highest to lowest order, upward alongside the white space until reaching the black module at the corner of the finder pattern. The bit that would otherwise go in that position is placed at column j = m-7, row j = 8, zero-indexing. Subsequent bits are placed to the right, with the last bit located at the last column j= m -1.

**Version Data**: The version data shows the version used. It takes 18 modules or pixels (3 wide by 6 high). A 3x6 duplicate is placed just above the bottom-right finder pattern. The encoding is mirrored diagonally along the NW to SE line. When building the encoder, the version data information is stored at the original row and column element, and the opposite flipped row and column indexes for the alternate. If the top-left corner element is $e_{00}$, what is located at module [x, y] in the original version data could be placed in module element [y, x] for the duplicate.

**Data and Error Correction**: after encoding the data an error correction algorithm is applied to ensure the symbol can be decoded even if partially obscured. The encoded data and the error correction information are stored in the free modules.

**Mandatory Blank Space:** of 4 modules in width are placed around the symbol, to prevent interference with other images or text.

## 2.6 QR Decoding

We discussed the split decoding problem in Chapter 4. Decoding was part of the implementation of decoding within the TrustZone while splitting the code between REE and TEE. Details and references are cited in Chapter 4.

## 2.7 Tokenization and Existing Protections Method

This wide range problem was cited many times in many places in the thesis. It represents the methods used to provide protections of transactions. Tokenization use divide between user merchant, and provider, other methods include the virtual credit card to reduce potential breaches of the databases. We have visited this briefly in Chapter 1 and discussed in detail in Chapters 5 and 6 and compared out method of Dynamic Offline TrustZone VCC generation. (DOT-VCC).

## 2.8 DOT-VCC Method

The DOT-VCC method utilizes credit card generation by bank to produce it offline by the user through an agreement with the bank. Its main difference with virtual credit card, is that it is timed, one time use, and many other combinations. The main point it is generates the number by the user. This removes many attacks surfaces. We have outlined this novel approach in the Chapters 5 and 6 in and the future work. All references and introductory information are outlined in Chapter 5.

# Chapter 3

# Assured Data Integrity of Camera and Location Devices

# Chapter 3: Assured Data Integrity of Camera and Location Devices

**Abstract.** Mobile phone devices constantly face new vulnerabilities from attackers to exploit. Many vulnerabilities allow attackers to gain full control over the operating system, and thus putting security critical operations at risk. Mobile payment systems are gaining more traction and security countermeasures cannot rely on operating systems for protection. ARM TrustZone provides hardware-based security, which is often used to protect key operations. In this chapter, we extended TrustZone functionality to offer robust security measures for specific I/O peripherals, namely, camera and location, to any application on demand. The design ensures integrity of data retrieved by the peripherals. Applications that can utilize this functionality include merchant-presented QR payment systems, location attestation for payments and other applications. The work is designed to offer seamless integration for application developers, and transparency to end-users. We demonstrated functionality on custom and modified existing applications. The added overhead is within expected margins. The work provides a feasible design for industrial implementations, where the vendors' installed services do not need coordination with application developers, and that offers flexibility for both vendors and developers.

## 3.1 Introduction

In this chapter and article [7], we discuss the development of a configuration that serves third party applications with no prior coordination with vendors, while maintaining peripheral data integrity. We have worked on two services, camera-scanned Quick Response (QR) codes that include payment information presented by merchants, and location services utilized by merchants to attest location of devices prior to transactions.

The solutions of this chapter maintain the integrity of merchant-presented QR codes and user location data for important operations. In section 3.2 we discuss background and related work, in section 3.3 we discuss the problem statement and threat model. Sections 3.4 and 3.5 present our design and implementation, respectively. Section 3.6 presents evaluation and security analysis. Finally, we conclude and present some recommendations in section 3.7.

The problem main theme is data integrity. The normal world (i.e., Android) is free to access the data, but any data exchange with servers must contain the correct data. The task is to develop a product that is feasible for industrial implementation, with no need for vendor coordination with application developers when implementing the services. For other background work, refer to references [6] [44]. Related background to TrustZone is covered in Chapter 2 section 2.3.

## 3.2 Background and Related Work

### 3.2.1 I/O Trust Access and Data Integrity

This activity includes many operations that use peripherals for certain functions. Sometimes they require accuracy and data integrity, in others they need confidentiality. We present here two examples of services we have developed to protect under TrustZone operations. They are namely using camera, and location attestation.

### 3.2.2 QR Merchant and Buyer Payments

1) Buyer-presented. Requires users to present generated QR codes for merchant scanners. These codes contain sensitive information about the buyer and hence data secrecy is essential, and that was addressed by [2]. Another stronger method we have developed is covered in Chapters 6-7.

2) Merchant-presented. Merchants present QR codes containing merchant information for buyers to scan and complete payments. Merchant QR information does not require secrecy, but integrity is crucial to ensure the payment goes to the intended destination. Merchant-presented QR

includes static and dynamic QR codes. Static QR does not include the payment value, while dynamic QR includes it. Figure 3.1 shows the two QR payments methods.

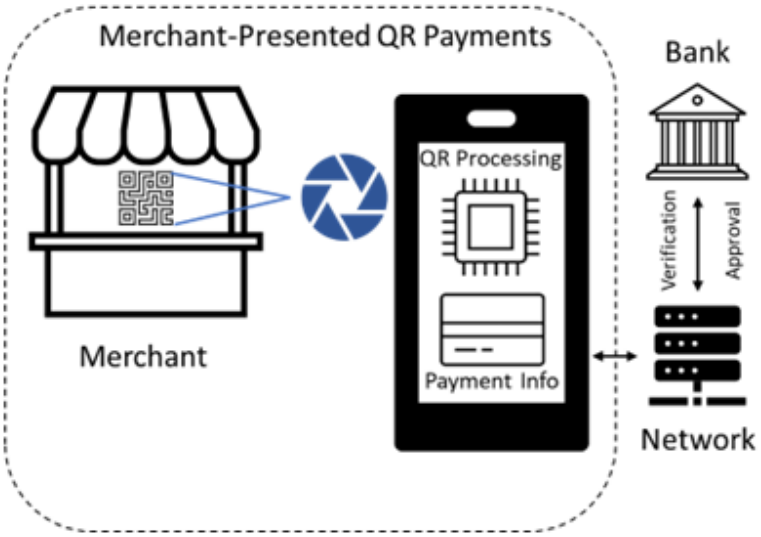



Figure 3.1 Merchant presented QR payments.

### 3.2.3 Location Attestation

Advanced mobile payment systems utilize location services to attest buyers' locations during transactions. This attestation is needed to protect buyers from unauthorized payments that can be launched by malicious activities. The location of buyers is not secret, but the integrity of the information is critical. Location attestation serves payment verification and can be used for inquiring services, including social media.

### 3.2.4 ARM Trusted Execution Environment (TEE)

After the introduction of TrustZone in section 2.3 and [3], [4], [16], we will be using the TrustZone and the modified form of TruZ-Droid extensively. The TrustZone is integrated into a System-on-Chip (SoC) design by vendors [3], [4]. The TrustZone hardware isolation includes the processor, cache, main memory, and other hardware. The hardware can be split physically or virtually to achieve the TEE and REE environments. Physical splitting is costly, given it dedicates hardware resources for TEE uses that are inaccessible to the REE. Virtual splitting, on

the other hand, allows both environments to share resources, but not simultaneously. That is, when TEE uses a hardware resource, the REE cannot access it for the whole duration. In both approaches, the TEE reserves the hardware, and the difference is in dynamic allocation. Either way, TEE can still access the system wide resources at any given moment if necessary, including peripherals. Mobile devices switch back and forth between TEE and REE depending on the use-case. TrustZone implementation requires a presence of a Secure-Monitor that handles context switching through REE triggered Secure-Monitor-Calls (SMCs), which immediately invoke the monitor, and then proceed to the TEE. Figure 2.4 shows the two environments. The Secure Monitor is built into the hardware and always operates under secure mode.

TEE operates like a normal OS by having a kernel and a user space. The user space contains Trusted Applications (TAs), which conduct services to the REE, e.g., fingerprint authentication. REE contains TA clients (sometimes called Client Applications) which correspond to TEE's TAs. An application in the REE typically makes a call to the TA client, which then calls the TEE driver (in the REE) that then triggers the SMC which invokes the necessary TA service.

### 3.2.5 Related Work

In this section we review some of the works closely related to our work as background information or parallel forms of activities. The more closely works will be reviewed here.

**Stefan Saroi et. al. [34]** propose on-off control of peripherals on smart devices as a key to security and privacy in many scenarios. They present (SeCloak), an ARM TrustZone-based solution that ensures reliable on-off control of peripherals even when the platform software is compromised. They designed a secure kernel that co-exists with software running on mobile devices without code modifications. An Android prototype demonstrates that mobile peripherals like radios, cameras, and microphones can be controlled reliably with a very small and trusted

computing base and minimal performance overhead. This service is very useful for controlling situations that require such services and the concept is utilized in our work. Peripheral control [6] and Visa location [35] solutions are also utilized in the work.

**S. Demesie Yalew, et. al. [36]** present the design for Android OS Integrity, namely TRUAPP which is a software authentication service that provides assurance of the authenticity and integrity of applications running on mobile devices. It provides such assurance, even if the operating system is compromised, by leveraging the ARM TrustZone hardware security extension. TRUAPP uses a set of techniques like static watermarking, dynamic watermarking, and cryptographic hashes to verify the integrity of the applications. The service was implemented in a hardware board that emulates a mobile device, which is used to conduct experimental evaluation of the service.

In addition, **S. Demesie Yalew et. al. [37]** present the design of T2DROID, a dynamic analyzer for Android that uses traces of Android API function calls and kernel syscalls, which are protected from malware by leveraging the ARM TrustZone security extension. They report the T2DROID achieved accuracy and precision of 0.98 and 0.99, for the two calls respectively, using a kNN classifier.

**Darius Suciu and Radu Sion [38]** present a framework for current ARM mobile devices that can detect applications control flow manipulation attempts, by examining the history of the executed control flow, that are altering instructions on the processor. The history examination provides information to implement fine-grained control policies, without additional binary instrumentation. Moreover, this framework can work with existing hardware and have a minimal impact on performance.

Regarding confidentiality of user's information, **Bo Zhao, et. al. [39]** propose a private user data protection mechanism in TrustZone to avoid risks. They added modules to the secure and normal worlds and authenticated the identity of Certificate Authority (CA) to prevent illegal access to private data. They have tested the system security, validity, performance, and reported that the method can perform effective identity recognition and CA control that protects the user private data. After adding the authentication modules, the separation time increases by about 0.16s, an acceptable cost for the improved security.

**Xianyi Zheng, et. al. [40]** propose a mobile payment architecture, employing trusted computing on an ARM TrustZone hardware isolation platform, which can ensure the transactions data security, realize privacy friendly payment, and provide trusted computing services to the system. They have implemented a prototype on a simulation environment by using ARM FastModel and Open Virtualization software stack for ARM TrustZone and presented an implementation on a real board by using ARM CoreTile Express A9x4. The security analysis shows the scheme can meet the security requirements of a practical m-payment with acceptable performance.

For Trusted Operations on Sensor Data, **Hassaan Janjua, et. al. [41]** present the design and implementation of a framework that allows capture of trusted data from external and internal sensors on a mobile phone. In addition, they have addressed the development of trusted operations on sensor data while suggesting a mechanism to perform predefined operations on the data while the trust is maintained. Their evaluation shows that the proposed system can ensure security and integrity of the sensor data with minimal performance overhead. The framework is not tested on real implementations and does not provide solid testing of specific operations. **Burke, Jeffrey et. al. [42]** introduce the concept of participatory sensing, in cellular phones, to

form interactive, sensor networks that enable users to share information. An initial architecture to enhance data credibility, quality, privacy and sharing in such networks is described.

**Liu, H. et. al. [43]** propose two software abstractions for offering trusted sensors to mobile applications. They present the design and implementation of the abstractions on both x86 and ARM platforms. They have implemented a trusted GPS sensor on both platforms and provided privacy control for a trusted location using differential privacy.

**Kamble, P.A. and Neha Patil [44]** discussed using TrustZone for payments systems. The paper introduces utilizing arrangement of cloud framework with the assistance of Raspberry Pi, claiming that would protect client information. The strategy applies consolidated use of steganography and optic cryptography for this logic. The issue with cloud-based arrangements is that servers are profoundly open through the Internet and hence extensively presented to programmers and malware. They propose the idea of Darkroom, a secured picture preparing administration for the cloud utilizing ARM TrustZone innovation. This framework empowers clients to safely handle picture information in a protected domain that anticipates presentation of delicate images.

**Waqar Anwar [45]** for mobile payments using Near Field Communication (NFC), a Secure Element (SE) is the preferred place to securely store cardholder data. The paper summarizes shortcomings in Global Platform's (GP) SE access control specifications, and weaknesses in its implementation by the Android Operating System (OS). Moreover, a coherent model for an alternate and secure SE access control is proposed using SE and Mobile Trusted Module (MTM) specifications. This new model is secured by design and can be implemented using existing specifications, technologies and hardware. We will consult with this work in our designs and implementations.

**S. Rehman and J. Coughlan [46]** present an implementation of a mobile payment using NFC. It is not clear if the implementation is actually working or can be applied to real time situations.

### 3.3 Problem Statement & Threat Model

How to maintain the integrity of camera and location data, for third-party applications, under compromised REE operating system, while using the protection of TrustZone and maintaining transparency to developers and users. The work assumes the availability of secure and trusted TEE. In addition, how to provide a solution that is feasible for industrial implementations with no prior coordination between vendors and developers. **The threat model** assumes that the Rich Executive Environment (REE) is vulnerable, and attackers can gain full access. This access can modify data flow in any application and can initiate unwanted operations. Mobile users prefer ease-of-use provided by their devices, and when they intend to make payments, QR scanning is convenient. With no security in place, attackers with access to REE can exploit users in 2 ways:

**1- Initiating false payments**: attackers can start-over a payment transaction using the exploited user device and provide the necessary information to proceed with unauthorized payments. For example, they can provide QR code and trick the users that they have scanned those codes. Furthermore, they can change the device location to make the transactions match their own geolocations to process their own transactions.

**2- Modifying ongoing payments**: the users initiate payments on their own, and when they are scanning the QR codes, attackers can change and manipulate the information to alter the destination and the payment amount. **The attack surfaces** can occur during data retrieval from the peripheral, during data processing, or before sending data to the server. Figure 3.2 shows the details of the attack surfaces for the two attacks possibilities.

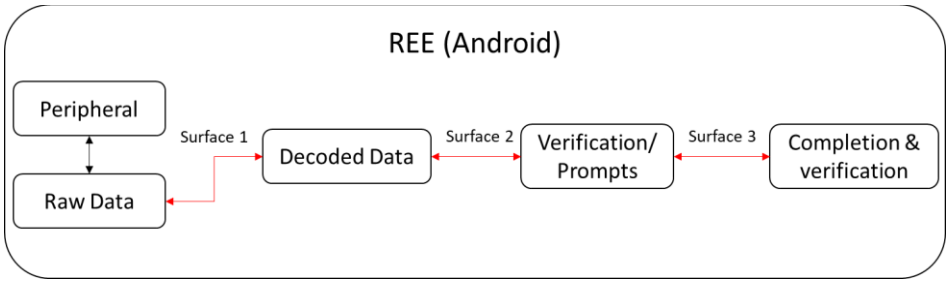



Figure 3.2 Attacks surfaces for QR transactions under a compromised REE.

The payment problem can be addressed in the TrustZone as follows: 1) Obtain trustworthy data from peripherals, 2) Provide the needed data and attestation to servers, 3) Transparency to application developers & users, and 4) Maintaining a small TCB size.

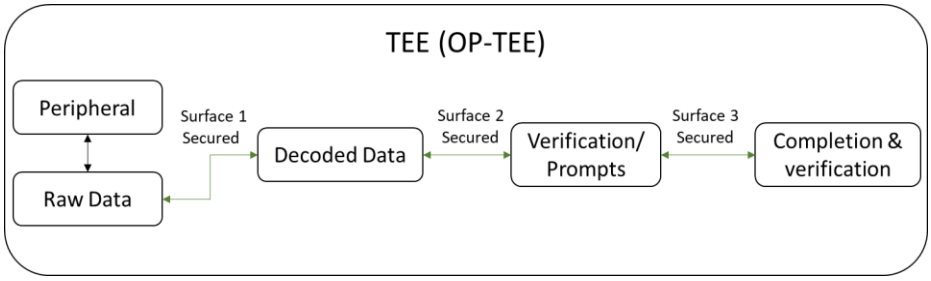



Figure 3.3 Securing transactions through moving functions to TEE.

Since the work assumes security and availability of the TEE, attacks that target the TEE or deny it from servicing REE are out-of-scope of this work. Figure 3.3 shows the general configuration of the operation by moving the necessary parts to TEE.

## 3.4 Solution Design

### 3.4.1 Data Retrieval from Peripheral Devices

The TrustZone designs allow for exclusive TEE-only access to all system resources including peripherals, hence it is crucial to block REE access to the peripherals during data retrieval. In our case, we target the camera and geo-location devices. Therefore, the design relies on developing a trusted service application (TA) that collects data from each device. Furthermore, we need a corresponding TA client in the REE to trigger the necessary calls to invoke that TA. Figure 3.4 shows access to peripherals under our design. It is important to maintain accessibility of the

peripheral for normal REE operations and only block it when TEE needs the data. Getting data from the camera and geolocation are not the only tasks needed.

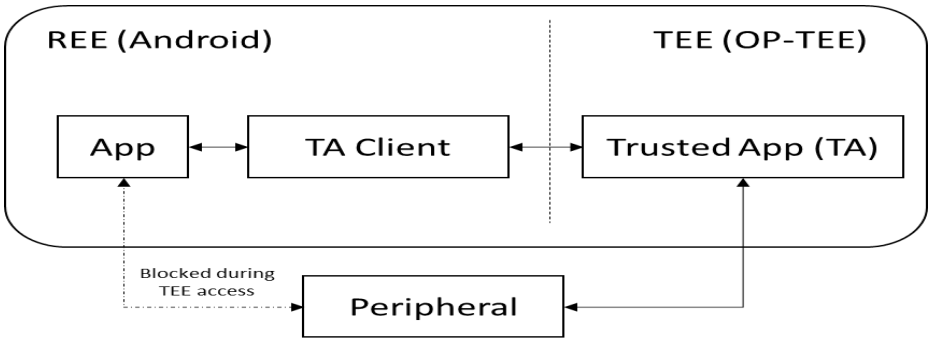



Figure 3.4 Peripheral device access for the Rich (REE) and Secure (TEE)

For example, static QR codes do not contain payment amounts. Therefore, it is necessary for the TEE to get the amount from the user directly. The TruZ-Droid solution provides means to capture the input from an on-screen keyboard which we incorporated into this design [2].

### 3.4.2 Data Processing

Data from the peripherals comes in all forms and sizes. In our case, we are dealing with camera images and geolocation data. Camera peripheral devices usually contain an Image-Signal-Processor (ISP) which configures the camera and formats the data as desired. The data can either be in RAW, bitmap, or other image formats. Geolocation peripherals use NMEA statements as standard. The NMEA statements describe information, including location coordinates, movement speed, direction… etc.

Since we need to maintain data integrity, it is not allowed to delegate data processing to the REE because of the assumed compromise. Therefore, all data processing is executed within the TEE. We decided to include this functionality within the same TA that retrieves the data from the corresponding peripheral. For NMEA statements, this procedure is straightforward, because they can easily be decoded into latitude and longitude. Camera data is a bit trickier. QR processing is not algorithmically difficult, but it requires a relatively large code base given the limited library

support from TEE. A tradeoff is needed: 1) process the QR inside TEE and carry the entire extra code base inside the TCB, which adds burden on the TEE security and performance, 2) transfer the image data to the server without processing.

The first approach has many advantages especially in the case of static-QR codes, when the QR code does not contain the payment amount. This enables TEE to immediately recognize the situation and ask the user to input the amount.

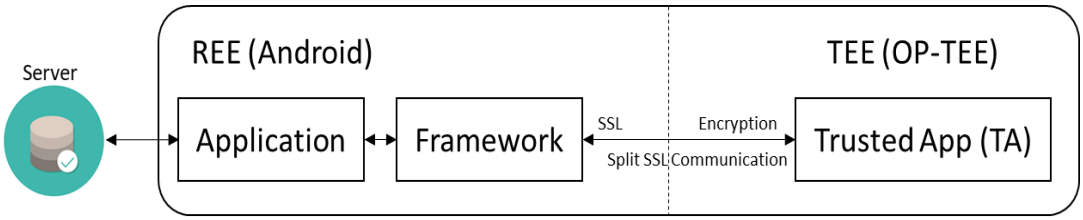



Figure 3.5 Using Split-SSL secure communication with the server.

The second approach maintains a small TCB size, with a disadvantage especially if the user must enter the amount. This adds a significant performance overhead due to increased communication with the server over the internet. Still, we decided to use the second approach to maintain small TCB size. Finally, we need to send the data to the server securely. We decided to use TruZ-Droid's Split-SSL as shown in Figure 3.5.

### 3.4.3 Transparency and Seamless Integration

The services we provide in the TEE should be easily accessible to third-party applications. To ensure such seamless integration for third-party applications developers, we designed a special API that is integrated into the Android framework and requests the service from the TEE. Therefore, developers will just need to change few calls in their applications to utilize our provided services without diving into any low-level code.

Our design considers maintaining transparency to the user. If a service is too imposing on the users, they may not use it. Therefore, our design maintains the same procedural flow while

ensuring data integrity. Capturing the user input for static QR codes should have an indicator letting the user know when they are operating under secure environment. This can be achieved using an indicator peripheral (e.g., LED). It is important that this peripheral use is restricted to TEE and never accessible for REE.

## 3.5 Implementation

Commercial devices do not typically come ready for development. Even doing basic tasks such as rooting the device requires unlocking the bootloader which can be involved and lengthy process. Therefore, developing this design on commercial devices is not feasible. Meanwhile, Hi-Key provides a set of development boards that are suited for this purpose. We chose Hi-Key 620 board given it both has Cortex-A series cores, which are used in commercial Android devices, and it officially supports OP-TEE. The OP-TEE support comes combined with Android Open-Source Project (AOSP) which gives a lot of control over development. There were many challenges to overcome in the development. One major issue when working on TrustZone is the lack of driver support. This is usually not an issue for vendors who make their own equipment. Therefore, we had to work our way around this issue. OP-TEE comes pre-equipped with serial UART drivers, so we picked UART compatible camera and GPS peripherals. Figure 3.6 shows the layout.

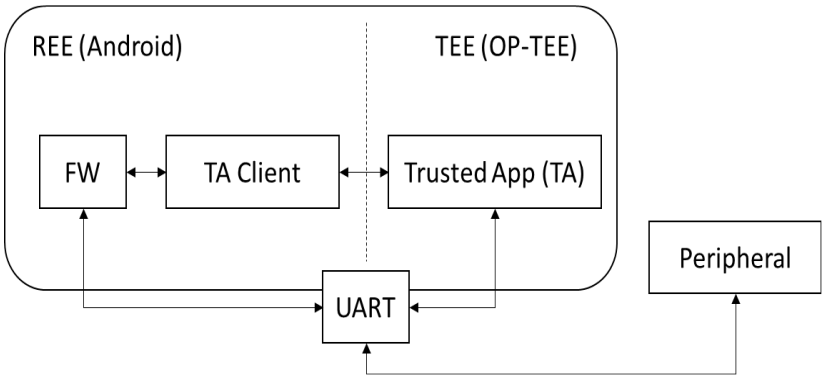



Figure 3.6 Connecting peripheral devices to Hi-Key board.

The UART camera device still required writing a second-level driver to synchronize, configure and capture photos. As for the GPS device, it was more straightforward given it sent ready for use NMEA statements. We wrote one service TA for each peripheral as they provide different services.

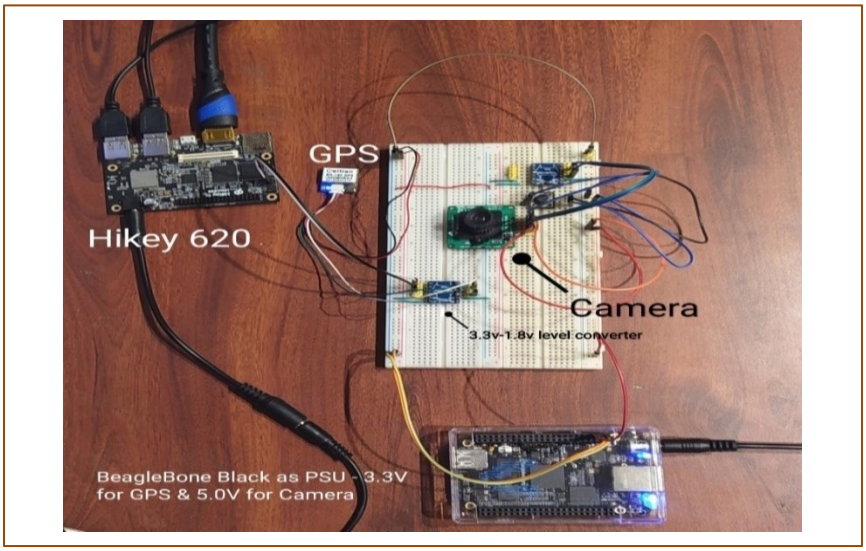



Figure 3.7 Connected circuitry of the system.

Figure 3.7 shows the connected circuitry of the system, and Figure 3.8 shows the system component devices. Once having the drivers functional, we developed the data processing portion. The challenge is to maintain a small TCB. That was difficult, because OP-TEE does not support most of the standard C library found on Android or Linux based systems. Any addition of standard C libraries also counts as addition to the TCB and might put the TEE at risk by introducing vulnerabilities to the code base.

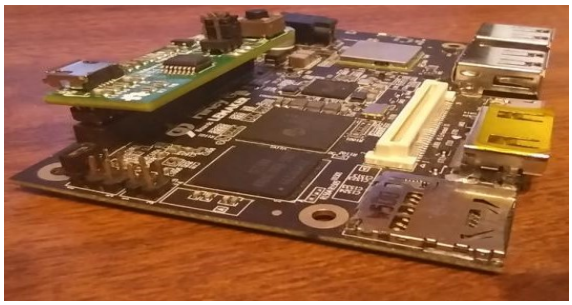
This was a reason we chose delegate decoding to the server despite the added overhead. Overhead is not of high concern regarding performance. This is mainly due to the nature of the services as they are typically low frequency use-cases. Still, we developed a measure of the overhead when using TEE versus REE use only as discussed in the next section.

**UART GPS sensor**

**UART camera**


**UART volt converter 5v → 3.3v**

**debugging kit for hi-key 620**



**UART Hi-Key 620 main board**

Figure 3.8 System configuration hardware components.

The final challenge was to provide seamless integration for developers by minimizing the number of changes needed to utilize the TEE functions. The modifications include Android framework in several ways. We have also developed modified APIs to provide that can be used by the developers as a replacement to their APIs.

Figure 3.9 shows the path an application takes to invoke TEE services. The TEE path (TA) is given by: App → TEE-API → TEE Framework→ TEE Daemon→ TA-Client→ TA → Peripheral and the REE application path is given by App→ REE API→ REE FW→ Peripheral.

Regarding Split-SSL and TruZ-UI, we ultimately decided to simulate their service because implementation does not add value to the proof-of-concept. To ensure the simulation does not affect the evaluation results of our service, we artificially added the Split-SSL and TruZ-Droid overhead so it would count towards the total overhead of our service. Outside of the overhead

values, those services behave more like black-boxes with respect to our service. In essence, Split-SSL is the black box necessary to securely transfer data to the server and ensure the server is aware the data received is correct. And TruZ-Droid is a black box which allows for the secure retrieval of the amount necessary for the payment processing. With that in mind and outside of the overhead, the remaining details of those services do not affect our evaluation results.

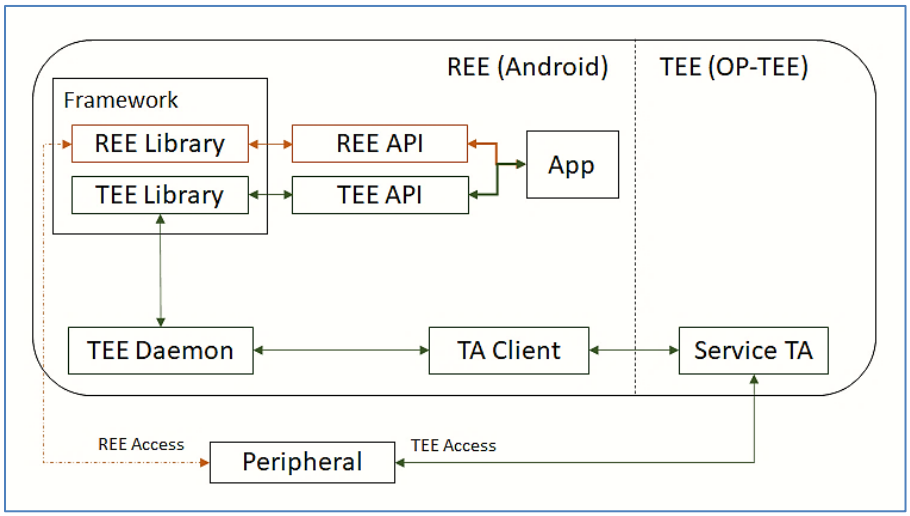



Figure 3.9 Application invocation with TEE and REE services.

Finally, the implementation also required changes to the server-side to accommodate the existence of TrustZone and its attestation. This lets the server know the integrity of the data. The changes include having the server decode the QR image and communicate accordingly. And the necessary changes to simulate Split-SSL as described in [6]. Table 3.1 shows results regarding overhead increase due to using TEE vs. REE functionalities to complete the operation for the two services. They are denoted as Base Location and Base Camera. The Overhead = (TEE time – REE time)/REE time. The results show the added overhead compared to an operation that does not utilize the TEE to retrieve and send data from the peripherals to a remote trusted server. In other words, the overhead is the cost of the security added by our services. Figure 3.10 shows the sequence diagram beginning from user initiation of payment request until the final confirmation.

Table 3.1 Results of the overhead for the two base services. 100% for Base Location.

| Application TEE path | Relative TEE Time % | Application REE path | Relative REE Time % | Overhead% |
|---|---|---|---|---|
| Base Location | 100 | Location REE Time | 85 | 18 |
| Base Camera | 120 | Camera REE Time | 95 | 26 |

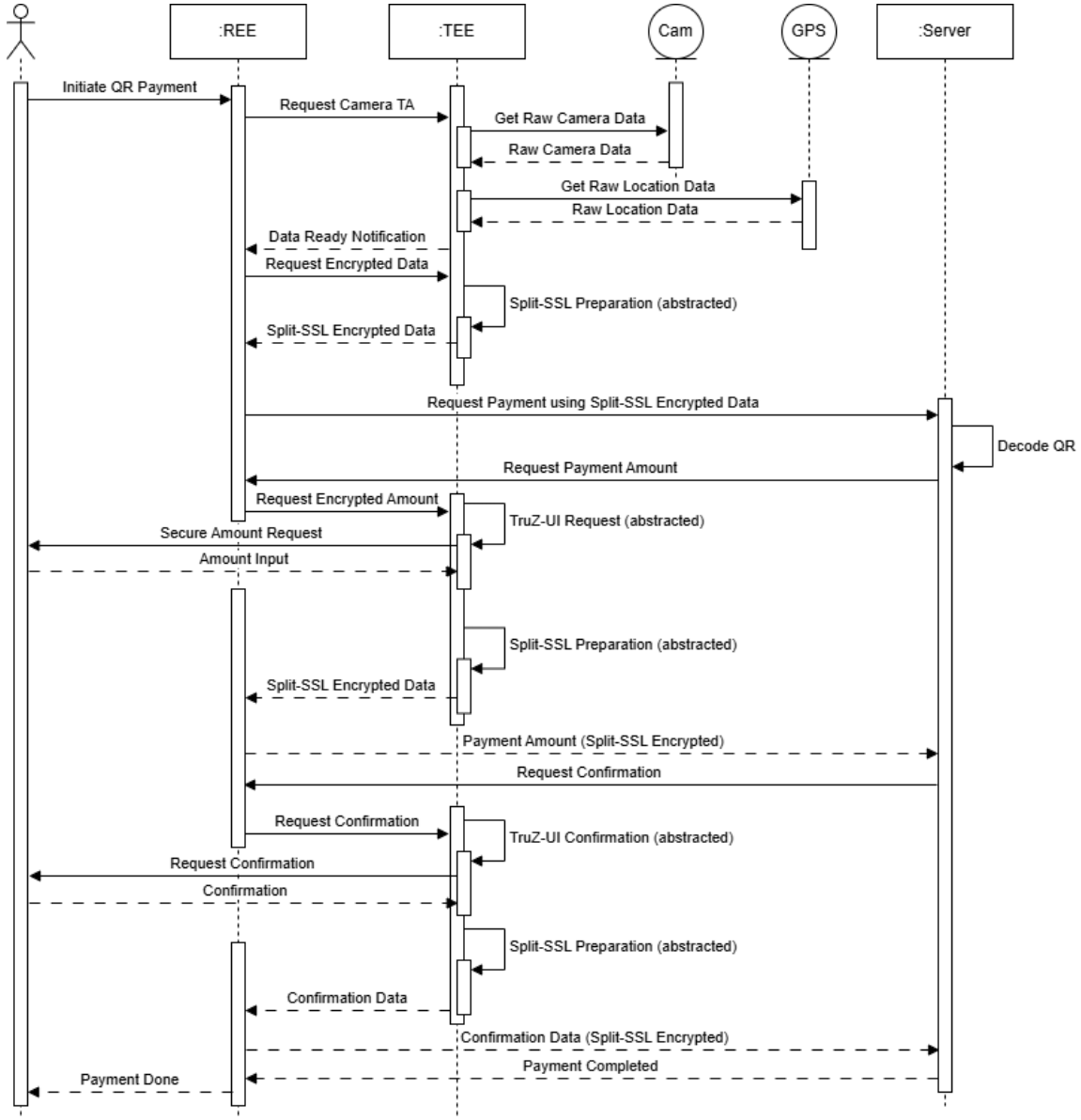



Figure 3.10 Sequence diagram for static QR code processing under system design and implementation

## 3.6 Evaluation & Analysis

To demonstrate functionality, we have performed the evaluation in two stages: namely, mock and existing applications. Both tests required our custom-built server to evaluate if everything is under the controlled environment.

### 3.6.1 Mock Applications

Mock applications are custom built to test correctness and functionality of our provided utilities. We have created an application to test QR scans, and another one to test location services. Both applications support non-TEE and TEE functions to measure performance overhead.

### 3.6.2 Modified Applications

We have downloaded a list of existing applications and modified their API calls to demonstrate utility on real-world examples, and equally important, to demonstrate seamless integration for developers. Naturally, we redirected their calls to our custom server. Some of the external applications provide their source-code online, while others are closed-source. The latter category needs repackaging attack to make the necessary changes. This adds difficulty given that we no longer work with source-code, but the Dalvik software instead. Furthermore, many modern applications rely on pinned server certificates to ensure they are not redirected by outside malicious activity, we also had to override this security mechanism for completeness. Accordingly, we did not implement external applications but developed our base applications to prove the feasibility. The modified applications and change in lines of code is shown Table 3.2.

Table 3.2 Applications, services and lines of code needed.

| Application | Service | LoCs App + API | TA LoCs in TCB |
|---|---|---|---|
| Base Location | Location | 1000 | 150 |
| Base Camera | Camera | 1500 | 300 |

The application-API change is what one would need to modify the existing application. In our case it is the base application we developed. The TA LoCs in TCB is the needed code to install inside the TrustZone and in this we care to make it minimum.

### 3.6.3 Evaluation Results

Table 3.3 shows the assessment of integrating the service with the intended applications in terms of data integrity success and performance overhead. The test of data integrity means assurance of no errors allowed in the data flow or readout. This is quantified by comparing the data coming from the peripherals into the TEE with the data received by the remote server. With the help of Split-SSL, we ensure the server knows it has data that was attested by the TEE. And while it is not possible to guarantee the data is received by the server (due to possible DoS attacks), we can guarantee that the server can verify the integrity of any data it receives. In other words, if the server receives fake data from the REE, that data cannot carry the attestation of the TEE, so the server knows the data is not trustworthy. On the other hand, if the server receives data attested by the TEE, it knows the data is trustworthy, and therefore can complete the transaction.

Table 3.3 Assessment of the applications integration and performance.

| Application | Service | Implementation Success | Data Integrity | Performance Overhead% |
|---|---|---|---|---|
| Base Location | location | yes | yes | 18 |
| Base Camera | QR payment | yes | yes | 26 |

The performance overhead is measured in terms of percentage of change in the application performance with and without invoking the intended TEE services.

The overhead average for secure QR payment processing was around 26%, the variations come largely from the type of QR processing involved. For static QR codes (where the user needs to manually enter an amount) the overhead is higher compared to dynamic QR codes that include al information. We address this in more depth in Chapter 4 as we discuss the QR decoding problem which aims to securely process images to minimize the amount of data sent over Split-SSL to reduce the total overhead of employing our design for secure QR payment processing.

### 3.6.4 Analysis

Our analysis was focused on integrity of data retrieval, amount of added Lines-of-code (LoCs) to the TCB, and performance overhead increase.

ARM TrustZone provides restricted hardware access to system-wide resources when operating in secure mode (or under TEE) [4]. The UART lines are no exception. Given we are not concerned about data secrecy, both camera and location peripherals are available for REE at any given moment except when TEE requests data from them. This is ensured by the TrustZone implementation, which in turn ensures the integrity of data captured by TAs from the peripherals. The secure transfer of the data to the server and the secure retrieval of data from the UI are problems that were addressed by Kailiang et. al. and the key point is we can guarantee, from their work, that the server is aware whether it is dealing with trustworthy data as explained in their analysis of Split-SSL and TruZ-UI [2]. Therefore, the utilization of Split-SSL and TruZ-UI designs is sufficient for data integrity which is a primary focus of our design.

As explained before, maintaining small TCB size increases verifiability of TCB security. Therefore, it's important to count the amount of code added to the TCB. Table 3.4 showcases our added LoCs. The camera TA mostly consists of the UART camera driver, and this varies heavily in commercial applications due to different devices and driver implementations. Despite that, the

number of LoCs added is relatively small. As for the location TA, smaller given it is mainly decoding NMEA messages.

Table 3.4 Showcase of lines of code added to the TCB for trusted applications.

| Trusted Application | LoCs added to TCB |
|---|---|
| Camera TA | 300 |
| Location TA | 150 |

Finally, performance overhead is within the expected margins, since context switching using SMCs is an expensive process. We did notice a significant increase in static-QR applications due to the back-and-forth communication with the server, this is expected as discussed in the design where we cut performance to maintain smaller TCB. Adding QR decoder in the TEE would reduce this overhead but increase the TCB size.

## 3.7 Conclusions

This work demonstrates how TrustZone can be extended to offer robust services to third-party applications without the need for developers to coordinate with vendors. We have built a system that ensures integrity of data collected from peripherals, camera and location devices, and maintained the integrity all the way to the server. The added overhead is acceptable when compared to normal untrusted operation. The added security does not introduce inconveniences for user interaction given the transparency of flow, nor does it create inconvenience for applications developers by requiring minimum changes to accommodate the provided TEE functionality. Our work provides means for extensions towards other peripherals in similar manners. It is feasible to secure Near-Field-Communication (NFC). The design offers seamless integration for developers as well as transparency for user experience. Table 3.5 highlights the key points and differences between our design and other relevant designs with emphasis on the overhead and total added LoCs to the TCB.

Table 3.5 Design highlights and comparison with other works summary

| Design | Highlights | Comparison to our design |
|---|---|---|
| Our design | • Addresses peripheral data integrity<br>• Offers secure path from peripheral to remote server<br>• Provides attestation of the data<br>• Processes QR payments<br>• Transparent security to users<br>• Minimal modifications for developers<br>• Small TCB additions | N/A |
| SeCloak [34] | • Addresses peripheral control<br>• Offers minimal and secure on/off control over peripherals<br>• Transparent privacy control to both users and developers<br>• Small TCB additions | Does not offer data integrity assurances and cannot protect QR-based transactions |
| TRUAPP [36] | • Addresses REE application integrity<br>• Utilizes TEE to detect static/dynamic manipulation of apps | Does not offer data integrity assurances and cannot protect QR-based transactions |
| T2Droid [37] | • Addresses Android API calls integrity<br>• Uses dynamic analysis to detect malware in the calls<br>• Useful for debugging | Does not offer data integrity assurances and cannot protect QR-based transactions |
| Xianyi Zheng, et. al. [40] | • Addresses mobile payment security<br>• Offers secure monitor and I/O access<br>• Offers secure storage for financial data | Does not offer data integrity assurances and cannot protect QR-based transactions |
| Hassaan Janjua, et. al. [41] | • Addresses secure data retrieval from internal/external sensors<br>• Offers data attestation and secrecy for data sent to remote server | • Does not offer data integrity assurances<br>• Cannot protect QR-based transactions |
| Liu, H. et. al. [43] | • Addresses sensor data attestation and secrecy assurances<br>• Utilizes TEE to securely retrieve the data from the sensors<br>• Signs the data with private keys and sends it over the network<br>• Offers seamless integration for developers through abstraction | • Does not address QR payment processing path<br>• Requires large TCB additions to the TEE by porting OpenSSL library into the TEE |
| Kamble, P.A. and Neha Patil [44] | • Addresses online payment security using TrustZone<br>• Offers camera-based verification of pre-established online payment platforms that have ledgers<br>• Uses TrustZone to confirm online transactions | • Does not address the secure data path of the image data<br>• Cannot process QR payments<br>• Strict to online payments |

# Chapter 4

# Split-QR Decoder Hybrid Design for ARM TrustZone

# Chapter 4: Split-QR Decoder Hybrid Design for ARM TrustZone

**Abstract.** In Chapter 3 we have built a system that ensures data integrity from merchant-presented QR code transactions by designing a secure data path between the peripherals and the TEE, and then we used Split-SSL to transfer the data to the server. Decoding was delegated to the servers with some disadvantages regarding performance and convenience for application developers. In this chapter we have designed a novel hybrid method by splitting the QR decoder between the TEE and REE. We have compared three different methods: 1) full operation in REE, 2) Split-QR decoding with dynamic and static QR, and 3) server decoding under dynamic and static QR payments. The five settings showed the feasibility and advantages of using the Split-QR. Advantages compared to the server-based decoder case include significant performance improvement and increased convenience for developers, while adding manageable code to the TrustZone. The success of this application encourages the design of a generalized framework to use split operations, where the TrustZone performs the core critical operations, or delegates them to a dedicated server outside the system if the operations are too extensive. In addition, it manages the meta data that runs noncritical operations on REE. We envision a wider scope of services with large and complex tasks where the Android system cannot handle them generally.

## 4.1 Introduction

Quick Response (QR) code processing libraries exist in many programming languages. Under feature-rich environments, such decoders are not usually considered as complex systems. This, however, does not apply to Trusted Execution Environments (TEEs), which are based on small Trusted Computing Bases (TCBs) that require small amounts of code to maintain strict security analysis and keep the system with little-to-no vulnerabilities. Such TEEs are typically built using C code given its high versatility and low-level capabilities.

QR processing is not algorithmically difficult, but it requires a relatively large code base given the limited library support in TEEs. Their existence in TEEs can be crucial to achieve many operations more securely, e.g., processing merchant-presented QR payments. A simplistic engineering solution may opt to move the entire QR decoder library into the TEE and bring with it much of the standard C library. While such approach would serve the intended functionality, it can introduce numerous vulnerabilities into the TEE and hinder its purpose. To put things into perspective, the ZBar decoder library consists of nearly 31k Lines-of-Code (LoCs), while Global Platform's Open-source Portable TEE's (OP-TEE) entire TCB consists of ~270k LoCs. Factoring in all the necessary C library dependencies would add over 50k LoCs (~20% size increase) into the TEE for a single functionality which would be nearly impossible to analyze for security purposes. An alternative approach that maintains TCB's security is necessary.

In Chapter 3, we had two options to consider: 1) the simplistic engineering approach described earlier, or 2) delegating the decoding process to the remote server which processes the payment. We chose the latter to maintain small TCB size and security, given it is of an upmost importance. While the second approach maintains a small TCB size, it carries its own disadvantages. It adds a significant overhead due to increased communication with the server over the internet. This is magnified in the case of static QR codes, where the amount of payment is not embedded into the QR code, and further communication is necessary. This shortcoming is not present in the first approach, given the TEE can immediately recognize the situation and ask the user to input the amount before initiating communication with the remote server.

In this design, we introduce a novel third approach that carries the advantages of both approaches, namely, little overhead while maintaining small TCB. The idea is to split the QR decoder into two components: 1) core decoding functionality and 2) meta-data processing. The

solution is to rely on the Rich Execution Environment (REE) to process meta-data, while the TEE does the core decoding. Meta-data processing accounts for a large portion of the decoder function.

In addition, as presented in Chapter 3 and [7], we continued using TruZ-Droid's Split-SSL for secure data transfer to the server; keyboard integration for secure amount retrieval from the user; and UI confirmation for final attestation step. In addition to the descriptions of Chapter 2 regarding the TrustZone and the Android problems, we review here only the most relevant topics to the decoding problem we are addressing. Methods shared with the work of Chapter 3 will not be repeated.

In section 4.2 we review related work to decoding methods. In section 4.3 we preset the formal problem, and section 4.4 outlines our new design solution and compares it to the previous design. In section 4.5, we present the implementation of the design solution, and section 4.6 covers testing and evaluation of the implemented solution. Finally, section 4.7 outlines conclusions and extensions to new related problems.

## 4.2 Related Work

### 4.2.1 QR Decoding using ZBar Library

The library is about 31k LoC, and the decoder spans over 6k Lines of Code (LoCs). Adding the C library dependencies, the problem becomes quite challenging. Here we describe the basic functions of the library.

**ZBar C Source-code Library** [49], [53]. There are scores of commercial and shareware programs that read many different symbols, the best open-source readers available is the so called ZBar C-source code library. This library provides high performance, stable, robust library components with supporting infrastructure that makes it easy to use in a variety of applications.

We will use the ZBar library components to achieve our implementation in the TEE environment [3], [4], [48] [57].

The ZBar library was originally designed for Linux-based systems but was extended later to support other systems. It was also originally built to run on x86-based architectures, but now it supports ARM architecture. The library can be ported, and it works with Android's NDK [54]. For our purposes since we will be splitting the decoder between TEE and REE.

**ZBar Operation**. A common design for a barcode/QR image scanner is to apply digital image processing techniques to the images. Exact details vary, but this usually involves several filter steps to cleanup noise, sharpen and enhance contrast, edge detection and shape analysis to determine symbol location and orientation, etc. All of the processing stages require CPU and are sensitive to various filter configurations which are difficult for end-users to understand or setup.

The ZBar library method is similar to "wand and laser" scanners: linear (1D) bar codes can be decoded by a simple light sensor passing over the light and dark areas of a symbol. The ZBar implementation makes linear scan passes over an image, treating each pixel as a sample from a single light sensor. The data is scanned, decoded and assembled.

ZBar further abstracts this idea into a layered streaming model. Processing is separated into independent layers with interfaces, which can be used together or individually. In the following we present brief description of operations:

1. **Video input**: abstraction of a video device which produces a stream of images for scanning. The library has interfaces to video4linux (versions 1 and 2) [55].
2. **Output window**: simple abstraction of a display output window that can present a scanned image to the user and accept input in response. To maximize flexibility, the window may be

opened and owned by the library or attached to an application managed window embedded in a GUI. The library supports basic X11 interfaces (XVideo and XImage) [56].

3. **Image scanner**: makes scan passes over a two-dimensional image to produce a linear stream of intensity samples. The input images may come from the video input module, or any external image source (such as an image file output by a flatbed scanner or digital camera). This module also incorporates the optional inter-frame consistency heuristic applied to a video stream.

4. **Linear scanner**: scans a stream of abstract intensity samples to produce a "bar width" stream. The intensity samples could be pixel values from the built-in image scanner, or from an alternate external image scanner, or raw sensor samples from a "decoder-less" wand or laser sensor. The bars are detected and measured by applying basic 1D signal processing to the input sample stream.

5. **Decoder**: the decoder searches a stream of bar widths for recognizable patterns and produces a stream of completely decoded symbol data. The current release implements decoding for EAN-13, UPC-A, UPC-E, EAN-8, Code 128 and Code 39 symbology codes. Support is planned eventually for PDF-417 and EAN/UPC add-ons.

6. **Processor**: one potential drawback of a fully independent modular approach is that it can take some coding to tie all of the modules together, complicating simple applications. The high-level "processor" module connects all of the other modules to flexibly support many common uses. For example, this makes it easy to pop up a window (or not) and scan for bar codes from video or image sources with very few lines of code. The included sample applications: zbarcam and zbarimg are two examples of how this can be done (UTSL).

7. **Widgets**: for applications which already have a GUI, it does not always help to open a separate window for reading bar codes. To facilitate tighter integration between the reader and an existing GUI, the library also comes with ready-made "widgets" for various popular toolkits (currently Qt4, GTK+-2.0 and PyGTK2). The test programs in the distribution are good examples of how to use these widgets to incorporate a bar code reader widget into an application.
8. **Language interfaces**: with the performance sensitive image processing done in C, library wrappers for Perl and Python make building a bar code application simple and efficient.

### 4.2.2 ZBar C Library Utilization in the TrustZone

ZBar is an open-source C library designed to decode QR codes with many different formats. The total Lines of Code (LoCs) count in ZBar is over 31k but the decoder itself is around 6k LoCs. However, the library depends heavily on standard C libraries to support different functionalities. Such libraries do not exist in the OP-TEE system running on the TrustZone. The challenge resides in porting this library to the TrustZone without introducing new vulnerabilities that can be exploited by the compromised Android through the exchange between the two worlds. Furthermore, ZBar also contains image processing code necessary to properly align the image to make the QR readable and ready for the decoder. This adds an extra significant amount of code to include within the TEE.

The LoC count isn't the only thing that matters when including source-code into the TEE. What truly matters is the security and verifiability of this code. As long as no vulnerabilities are introduced, the amount of code added does not matter critically. However, with large code, verification of source code becomes very difficult hence it's preferred to have smaller bases. The library can be ported to Android, and it works with Android's NDK. For our purposes, we're

interested in splitting it between the TEE and Android (REE). The ARM support makes our job a bit easier.

The ZBar library consists of:

- **Decoder**: this is the core portion of the library. It has the image reading functionality, the alignment processing and the many different formats to process from different QR types.
- **Entry point**: how the program is run, it can either process a single existing image or take feed directly from a camera device.
- **Interfaces**: ZBar has Python, perl and GUI interfaces to make it accessible from these different environments.

### 4.2.3 Related Work

In this section we refer to the reference article we used as a metric, and we review briefly the Split-SSL. In addition to the ZBar library decoding we touch upon some literature related to other decoding algorithms.

We reported [7] extending TrustZone functionality to offer robust security measures for specific I/O peripherals, namely, camera and location, to any application on demand. The work mainly ensures integrity of data retrieved by the peripherals. Applications that can utilize this functionality include merchant-presented QR payment systems, location attestation for payments and other applications. The work is designed to offer seamless integration for application developers, and transparency to end users. We demonstrated functionality on custom and modified existing applications. The added overhead is within expected margins. The work provides a feasible design for industrial implementations, where the vendors' installed services do not need coordination with potential application developers, and that offers flexibility for both

vendors and developers. In this work we use the ZBar library in the TrustZone to decode QR code images instead of relying on the vendors servers.

**Brief basics of Split-SSL Utilization** [47]

A popular technique for reducing the bandwidth load on Web servers is to serve the content from proxies. Typically, these hosts are trusted by the clients and servers not to modify the data that they proxy. SSL splitting guarantees the integrity of data served from proxies without requiring changes to Web clients. Instead of relaying an insecure HTTP connection, an SSL splitting proxy simulates a normal Secure Sockets Layer (SSL) connection with the client by merging authentication records from the server with data records from a cache. This technique reduces the load on the server, while allowing the unmodified Web browser to verify that the data served from proxies is endorsed by the originating server.

**Jeng-An Lin and Chiou-Shann Fuh** [50] emphasize the challenges that face barcode technology in Automatic Identification and Data Capture (AIDC), and how to decode QR code images efficiently and accurately is a challenge. They outline a revision to traditional decoding procedures by proposing carefully designed preprocessing methods. The decoding procedure consists of image, QR code extraction, perspective transformation and resampling, and error correction. By these steps, we can recognize different types of QR code images. Results show that the method has better accuracy than Google open-source 1D/2D barcode image processing library Zxing-2.1. Moreover, they evaluate the execution time for different-size images and show the method can decode these images in real time.

**Nivedan Bhardwaj et. al.** [51] propose a color QR code decoding algorithm for mobile applications by considering the color profile of red, green and blue channels, respectively. They have developed a prototype by including various image processing techniques and the open

source Zxing library. The article presents an Android based color QR code decoder implementation and performance evaluation. Experimental results show a high success rate of the methodology implemented.

**Madeline J Schrier** [52] presents the first rigorous study of resolution requirements in camera-based barcode scanners. What is the resolution needed in the captured image to unambiguously decode a barcode? For simplicity, they consider the UPC barcode, which is widely used in retail and commerce. A UPC barcode consists of black and white bars of different widths. The widths of these bars encode a 12-digit number according to a look-up table. They show that the camera model can be completely determined by a set of parameters defining the bar width and the shift in the image. These parameters can be determined using features of the UPC symbology, and the knowledge of these parameters allows exact decoding. They showed that the two parameters can be recovered from the image data for narrowest bars larger than three-fourths a pixel and in some cases, only half a pixel. Extreme cases show that unique determination of the digit is possible in worst-case scenarios, even under the presence of noise. Problem and Objectives

In Chapter 3 design we solved the problem by delegating the QR image decoding to the server. That causes difficulties regarding seamless integration for developers. The problem we are targeting is integrating a relatively large codebase into the TrustZone while maintaining TrustZone security. The TrustZone does not need all decoding functions that do not threaten data integrity. We want to process data decoding within the TrustZone, by including the essential parts of the ZBar library that are related to decoding in the TrustZone. If all transaction information is included then we send the request to the server, otherwise we ask the user to provide the needed information. The assumptions for this design are the following: 1) Android is completely compromised and attackers have full root access over it. 2) The TrustZone offers

secure framework for development. And under those assumptions, we want to achieve data integrity guarantees while maintaining transparent user experience and seamless integration for developers. DoS attacks against TrustZone APIs are out of scope and are discussed in Chapter 7.

## 4.3 Solution Design (Split-QR Decoder)

### 4.3.1 Design Analysis and Comparison

Instead of delegating the decoding task to the server or completely embedding the decoder into the TrustZone, we decided to split the decoder. Split-QR decoder is a hybrid approach that uses REE to process the image meta-data, while the core of decoding (reading the code's data) is the only part moved to the TEE. This novel hybrid design reduces the overhead of delegating decoding to the server, and it requires much smaller increase in the TCB size compared to moving the whole decoder into the TEE. For reference, Figure 4.1 shows the steps taken in Chapter 3 design for static QR code transactions, and it carries a lot of overhead due to the back-and-forth between the server and the TEE through Split-SSL

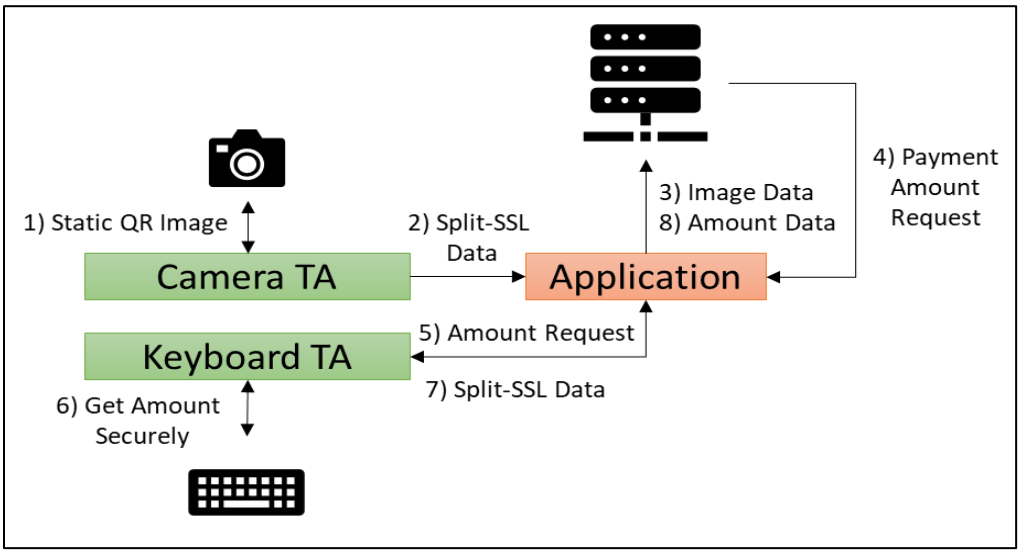



Figure 4.1 Static QR code payment initial server QR decoder design path.

One of the most apparent advantages in reducing overhead shows when processing static QR codes which do not include payment amount information. Figures 4.1 and 4.2 show the designs

for the server decoder and the Split-QR, respectively. The steps taken to get static QR code and amount are numbered and described in each figure. In Split-QR decoding, the server does not decode anything, and no payment amount request is necessary to initiate from the server side. The data is prepared using Split-QR and includes the amount as shown in Figure 4.2.

While the number of steps may seem larger, communication with the server is time-consuming. Sending images can be quite slow given it depends on the image size. This limitation is removed in the Split-QR design, where only text information (payment destination and amount) is sent to the server, and it is done at once which significantly reduces overhead. Other benefits include less changes on the server side.

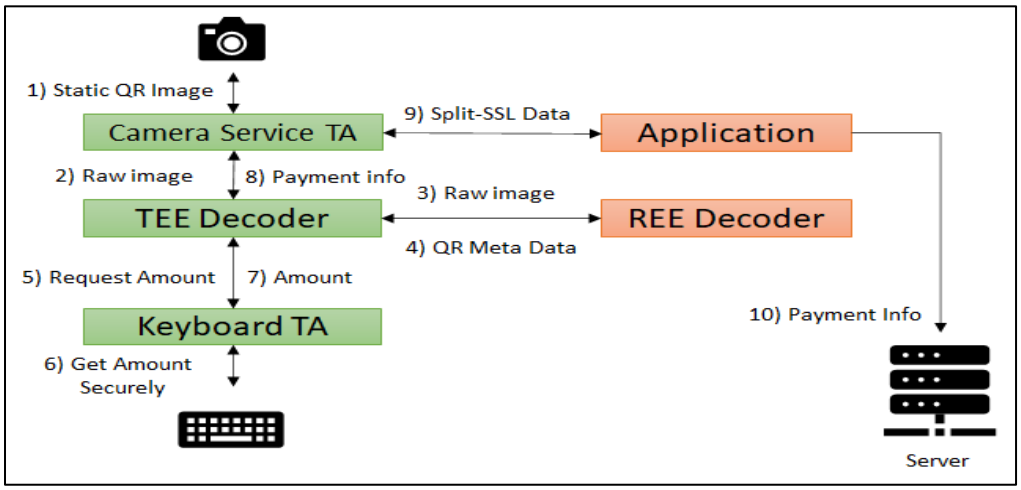



Figure 4.2 Static QR payment with TEE Split QR decoding design path.

In the Split-QR design, servers only need to implement TrustZone attestation mechanisms discussed in TruZ-Droid [2] approach. Furthermore, application developers will no longer need to implement extra functions to accommodate the extra communication introduced in the previous design.

It is important to note that REE decoder in Figure 4.2 is not directly called by the TEE decoder. Realistically, the REE API manages all of this when the application initiates the original request to process the QR payment. That is, the application calls the TA service, the image is captured

and transferred to the REE decoder through the API calls, the meta-data is sent back to the TA which decodes the code. The TA can automatically determine if the payment amount is not embedded in the code.

Therefore, it will automatically initiate a request to the keyboard TA (as described by TruZ-Droid) to securely retrieve the amount from the user. The data is then prepared and sent to the server for direct processed. Table 4.1 shows a summary of differences between the two designs.

Table 4.1 Summary of the differences in the designs.

| | Server QR Decoder | Split-QR Decoder |
|---|---|---|
| Overhead w.r.t REE-only operations | High | Low |
| Server modification | High | Low |
| TCB size increase | Small | Moderate |
| Seamless integration | Moderate | High |
| Transparency for users | High | High |

Figure 4.3 shows the sequence diagram for the Split-QR design which starts from the user request the amount and ends with the confirmation of the payment request. When compared to Figure 3.10, which shows the sequence diagram for the design from Chapter 3, we can see the server is not requested until the very end of the QR processing.

The TEE takes care of the processing completely internally without external interruptions outside the Split-QR exchanges. The server does not have to decode any QR codes, nor does it have to process a payment amount. All of that information is provided, at once, in the first request which saves communication overhead.

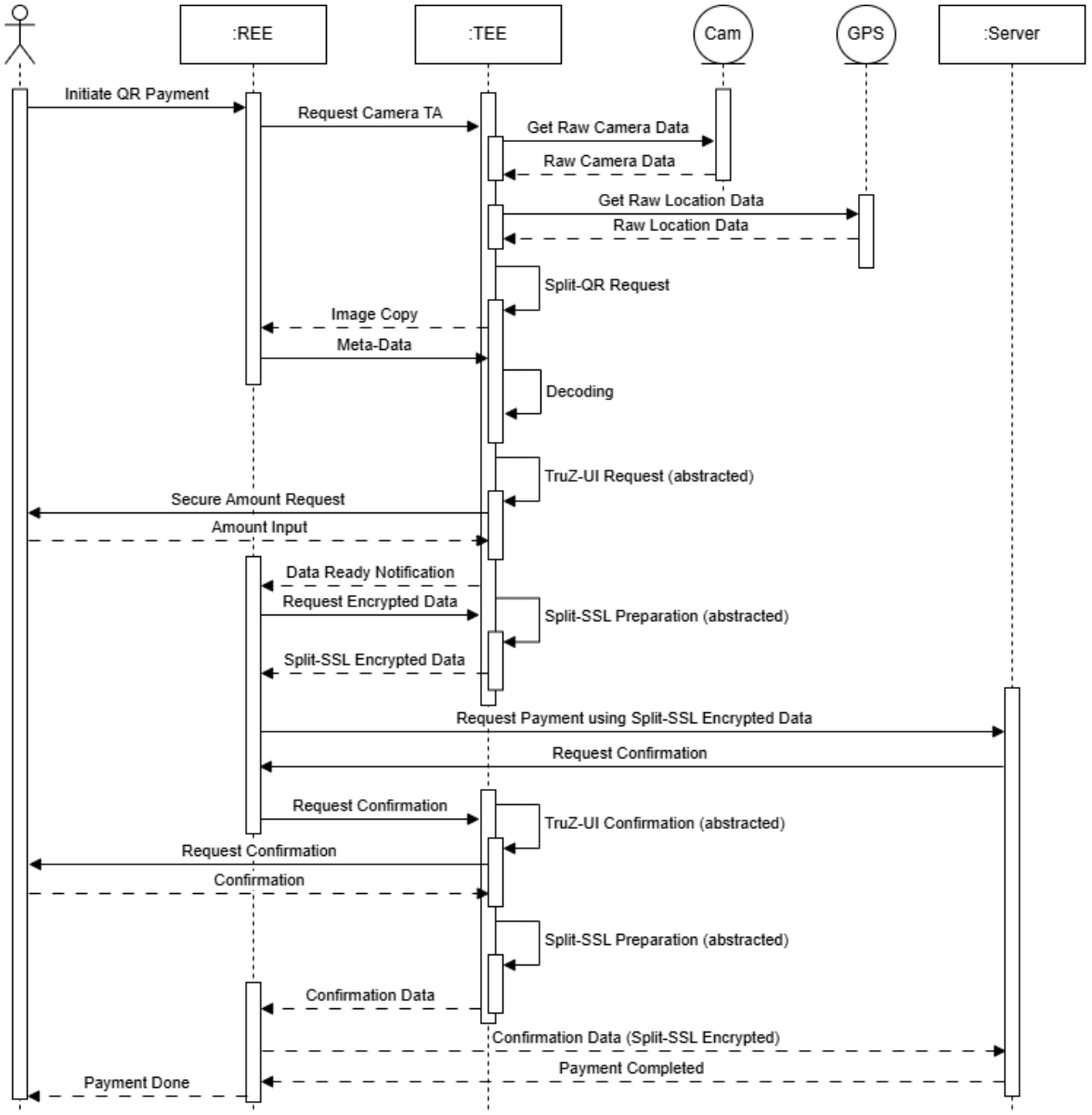



Figure 4.3 Sequence diagram for static QR payment processing under the Split-QR Decoder design

### 4.3.2 ZBar C Library Components in the TEE

ZBar decodes QR codes with different formats. The total Lines of Code (LoCs) count in ZBar is over 31k, but the portion corresponding to the decoder and its preprocessing is about 6k LoCs. However, the library depends heavily on standard C libraries to support different functions. Such libraries do not exist in the TEE OS. The challenge is to move only the needed parts to the TEE without introducing vulnerabilities that can be exploited by the compromised Android through

the exchange between the two worlds. ZBar contains the image processing code necessary to properly align the image to make the QR readable and ready for the decoder. Luckily, all of this part can operate in REE. It is important to minimize the LoC source-code included in the TEE. What is more important is the security and verifiability of this code. If no vulnerabilities are introduced, the amount of code added is not critical. However, with large code, verification of source code becomes difficult, hence it is preferred to have smaller bases. The TCB size increase is significant in this approach compared to the server approach, due to the library size. Table 4.2 shows the code partitioning where we moved the decoder's essential components to process the QR code without endangering data integrity.

Table 4.2 key components within the decoder which were partially moved to the TEE. The entire ZBar decoder library is much larger than the moved components.

| File | Total LoC | TEE | REE | No use |
|---|---|---|---|---|
| Sum | 6167 | 3717, 60% | 1468, 24.1% | 982, 15.9% |
| rs.c; rs.h | 820 (800;20) | 320 (300;20) | 0 | 500 |
| bch15_5.c; bch15_5.h | 200 (180;20) | 200 (180;20) | 0 | 0 |
| isaac.c  isaac.h | 180 | 180 | 0 | 0 |
| util.c util.h | 190 | 190 | 0 | 0 |
| binarize.c; binarize.h | 660 9640; 20) | 660 (640;20) | 0 | 0 |
| qrdec.h | 160 | 080 | 0 | 80 |
| qrdec.c | 3957 | 2087 | 1468 | 402 |
| Total LoCs 31330; TEE portion = 12% | | | | |

Therefore, the increase came up around 60% of the full decoder, but we did not need much from the rest of the library after removing 16% of the decoder for nonessential function and kept 34% in the REE where the operations there do not harm decoding data integrity. 3.7k LoCs is still within reason.

### 4.3.3 Splitting Criteria

Splitting the ZBar library between the two execution environments is not arbitrary, it is based on the following factors:

1. Minimization of external dependencies: the size of ZBar is not the only consideration due to the limitations in the TEE TCB as we do not have access to the vast majority of the standard C libraries that come with normal Android/Linux systems. The OP-TEE conforms only to the C99 standard and its corresponding libraries. Therefore, the portion of the library moved into the TEE must work with the available C99 libraries.
2. Integrity of extracted data: when relying on the compromised REE for meta-data extraction, it is important that the meta-data received from the REE does not lead to alteration of the data extracted from the QR image itself. The worst-case scenario should not yield more than a DoS attack which is equivalent to the REE not sending any meta-data to the TEE.

The aim is to target the security of both the TEE's TCB and the user operation's security. Those are the two primary areas of concern for this design, hence why the split was conducted. In the next section we discuss the scenarios an attacker may use to manipulate the meta-data sent to the TEE and what such attacks may yield.

### 4.3.4 Design Evaluation

In this section we discuss two major concerns for evaluation. The environment to use and the interaction between the Split-QR components in the REE and TEE.

**A. Environment**

In Chapter 3, we have tested the design feasibility on an actual ARM TrustZone capable hardware. We used Hi-Key 620 development board and OP-TEE for TEE development. Evaluation was done accordingly, and our tests and results covered integrity, performance and LoCs necessary to integrate the design into existing systems . The design in this work, however,

is only concerned with the decoder part, given the rest is already done. Proving feasibility and functionality can be simplified since development on the actual hardware is quite complicated and not necessary for this purpose. For instance, many tedious hardware settings must be done, which consumes time and effort with no practical benefits. Therefore, we chose to test this design in a simulated environment where we test the flow of Split-QR between simulated REE and TEE environments. The purpose behind the simulation environment is to isolate testing between the designs presented in Chapter 3 and the Split-QR decoder design in order to study the performance differences without the interference of external services that may affect the results. The Split-QR design splits a software library running in the user space and does not rely on specific hardware compared to the design in Chapter 3. In fact, the hardware in both designs would be identical, therefore, when we simulate the hardware we can better study the differences between the two designs and have more accurate comparison results.

Since the decoder part is the one we truly care about, we chose x86 environment to test the split functionality. We designed processes that simulate environments and components. Then we tested the flow of information between them. One component that took a lot of development time in the previous work was the camera driver in the TEE. Given that already exists, we can simulate this functionality by using a file descriptor and read the images from local storage as the desired result is to acquire the image by the TEE. The process then transfers the image to the REE process which extracts meta-data from the image. The meta-data is transferred back to the TEE to decode the QR code. Once complete, we check if the payment amount is embedded or not and launch the simulated keyboard TA. We can simply take input from the user since the keyboard TA is already proven to work in the TruZ-Droid design. Finally, sending the data to the server using Split-SSL and the final attestation with the UI TA are also simulated. Simulation

worked well to address the problem, due to the nature of different independent components. Split-SSL, keyboard TA, UI TA and our previous work TAs are all completely independent from each other and are already proven to work. Therefore, simulating the roles does not hinder the proof-of-concept we seek, and the design remains intact. In fact, it is more efficient to test the subtle variations. In this work we had time to fine analyze the codes nature and how their execution time is divided.

**B. Interaction Between Split-QR Components**

We have established that the TEE process transfers the image to the REE process, where preprocessing is made, and meta-data is extracted and sent back. However, it is important to show that this interaction does not hinder the integrity of the final data extracted in the TEE. The meta-data extracted by the REE contains information about the position and alignment of the QR code within the provided image. The REE conducts its analysis on a copy of the image not the original one. In other words, the image in the TEE remains untouched by the REE at all times, and that is ensured by the TrustZone implementation. Therefore, the meta-data is a description that helps the TEE extracts the image actual data. In this scenario, the worst a compromised REE can achieve is to provide invalid meta-data to the TEE decoder. This would result in invalid reading from the image, but it cannot produce a compromised reading of the data decoded from the actual image given that never changes. This sort of attack would fit more in the Denial-of-Service (DoS) category, and if the attacker wishes to conduct a DoS attack, there are much easier and more effective ways of achieving it. For example, the attacker can completely shut down the TEE driver in the REE which alienates TEE services. This type of attacks is extremely difficult to protect against, which is why we consider it out of scope. Nevertheless, the integrity of the data remains fully ensured, which is our main objective.

**C. Analysis of Potential Attacks by the REE**

Splitting code between the TEE and REE means we rely on the REE to process certain parts of the operation. By assumption, the functions performed by the REE are still subject to attacks from the remote attacker, which means those attacks have to be evaluated. The meta-data extracted from the image includes:

1. Position of the QR code and the skew angle within the image.
2. Positions of the finder, alignment, and timing patterns.
3. Content of the format data (error-correction level and mask pattern).
4. Content of the version data.

The position of the QR code within the image combined with the angle/skew of the QR code are the most important parts of the meta-data received from the REE. The REE uses the finder, alignment and timing patterns to find this information, and those patterns can, in turn, be extracted from the position/skew. Both types of information can be inferred from one another. Furthermore, the content of the format data (ECC level and mask pattern) and version data is purely supplementary and not essential for the TEE at this stage of decoding. Once the finder, alignment and timing patterns are available in the TEE, the module matrix of the QR code is possible to extract. The extraction is only performed on the authentic image received securely from the camera (as explained in Chapter 3) and produces a bit-matrix representing the QR code modules. The bit-matrix size depends on the version of the QR code, which is also embedded within the matrix itself (a sequence of 18 bits repeated twice in two fixed locations on the map). The data format bits are also embedded in the bit-matrix (15 bits repeated twice in two fixed locations as well). In essence, the verification of the information provided by the REE is an easy (straight forward and reliable) process. the difficult process that demands external dependencies is the one involved in finding the QR code within the captured image.

In terms of attack types performed by the REE, the attacks that can only yield denial of service do not represent a challenge to the configuration functionality. That is the case, because there are many ways to perform DoS attacks against the entirety of the TEE. For example, an attacker can completely shut down the TEE driver within the Android kernel, which renders any service calls requesting TEE services blocked. Therefore, an attack against the Split-QR decoder that only yields blocking of the service will be a wasteful effort that can be achieved with easier means. In this respect, DoS attacks against the TEE and its services are considered out-of-scope. With that in mind, there are two main categories of attacks in this system that are worth investigation, and each category has a range of associated attacks:

1. **Omission of meta-data going into the TEE**: the attacker performs attacks that aim to limit the amount of meta-data going into the TEE. This category of attacks includes removing parts or all the meta-data from the info sent back to the TEE.
2. **Manipulation of meta**-data going into the TEE: the attacker changes the meta-data before sending from the REE to the TEE after the REE extracts the data from the image copy. This category includes manipulation of parts or all the meta-data going into the TEE.

The attack scenarios can include one or both categories concurrently to achieve different purposes. In this analysis we will prove that such attacks will not yield decoding malicious data from the QR code inside the TEE. The main scenarios include:

1. Omission of all meta-data: in this attack the purpose is to block the TEE from decoding any data from the QR image. Such an attack represents DoS, but it does not pose a risk to the data integrity. This scenario also includes omission of all positional data (QR position, skew, and the patterns).

2. Omission of position and skew data: the attacker withholds the position and skew information from the TEE, while sending the remaining information (patterns, version data, and format data) as-is to the TEE. In such case, the patterns on their own are enough for the TEE to read the bit-matrix from the image. Therefore, the attacker does not achieve anything by omitting the position and skew data. But that can indicate a malicious activity on the part of the REE.
3. Omission of position and skew data with manipulation of the remaining data: the attacker aims to misguide the TEE by omitting the position of the QR code and skew information while changing the positions of the patterns. The finder pattern has very distinct shapes that must have blank space around them from all directions. This allows the TEE to verify if the pixels on the real image match the manipulated information received from the REE. If a part of the pattern was manipulated by the REE, then the TEE verification will discover that the information does not yield a valid QR code. Meanwhile, if all the pattern was altered, then the TEE verification will determine that the pattern does not really exist in the reported location, and the attack would yield no data extraction. As for the timer pattern, this pattern always connects to the finder pattern itself. Therefore, TEE verification will easily determine if the timer pattern is previously verified that there is a QR code using the finder pattern, which is an essential step anyway. The alignment pattern helps determine the rotation of the QR code, but it still has to coexist with the finder pattern properly. That means there can be no intersection between the two patterns, which is verifiable by the TEE. Finally, as explained previously, the version and data format information can be extracted from the bit-matrix, so manipulations of that information can be ignored upon failure of verification, without affecting the integrity of the data extraction.

4. Manipulation of position and skew data while omitting the remaining data: in this scenario, the attacker aims to manipulate the way the TEE reads the QR code from the image. Omission of the patterns means the TEE must read them itself from the position and skew information provided by the REE. In such an attack, the attacker can claim the QR code starts in a different location and claim there is a different skew to the image. If the claim suggests the QR code starts in a completely different area, then no patterns will be found and no QR code can be extracted, but it does not yield any manipulation of the data. Another case could have the attacker point to a place within the QR code itself aiming to take a portion of the real code to construct a different code during the bit-matrix reading. In such case, there is no portion within any QR code that can be interpreted as another QR code as we will outline shortly. Finally, if the skew is altered by the attacker, then the patterns (finder, alignment and timer) themselves will become self-contradictory. In other words, the TEE verification will determine that the resulting patterns cannot exist.

It is possible to make a scenario of every possible combination of omission and manipulation of the meta-data, but the scenarios described above cover all the cases implicitly. For example, any manipulation or omission of version and format data is useless for the attacker because they are found within the bit-matrix. In addition, manipulation attacks are addressed fully in scenarios 3 and 4 combined. The attacks cannot produce more than DoS due to the QR code structure itself. For instance, trying to claim that part of the alignment pattern is a finder pattern cannot produce a functional QR code as described by the standard. In such a case, it is not possible to have the three components of the alignment pattern with empty blank space around all three, and an empty space around their combination (the quiet zone). Furthermore, there is no possible QR code that will have a correct timing pattern between those three components of the alignment

pattern. This is primarily due to the masking pattern used to alter the data bits to make the QR code more readable for scanners. There are eight masking patterns, and all eight are evaluated in the encoding process to determine which one has the least penalty score (which is looks for different bit patterns in the QR code's data/ECC bits). The resulting effect of the masking pattern on the resulting QR bit-matrix is similar to salt-and-pepper noise effect. And it is used to remove scanners ambiguity when decoding, which includes a QR code embedded in another QR code. Table 4.3 summarizes the attack scenarios and their assessment.

Table 4.3 Summary of meta-data related attacks and their worst outcomes. In all cases data integrity is assured and malicious activity is discovered.

| Scenario | Attack type | Purpose of attack | Worst outcome |
|---|---|---|---|
| 1 | Omission of all meta-data | DoS | DoS |
| 2 | Omission of position/skew data | Misleading finding of QR code in image | Will not stop accurate decoding. |
| 3 | Omission of position/skew data. Manipulation of patterns. | Alteration of data read from QR code | No extraction or DoS |
| 4 | Omission of patterns. Manipulation of position/skew data | Alteration of data read from QR code | No extraction or DoS |

## 4.4 Implementation

As described in section 4.2.1, we chose to use x86 environment for our implementation to simplify the proof-of-concept. A major point of concern when it comes to simulating functionality on different architectures and platforms is to make sure it can be ported to the correct architecture, namely ARM architecture and its TEE. Another point of concern is to make sure the TEE process is actually completely independent from the development platform libraries (standard C library support). This step is necessary to ensure that the process does not have

uncounted external dependencies, which is crucial to prove the simulation is correct. The simulation environment we used is summarized in Table 4.4. The first concern can be addressed by removing any architecture-specific and platform-specific function calls in the decoder itself from the moment the image is retrieved until the data is extracted properly.

Table 4.4 Summary of simulation environment.

| | |
|---|---|
| Architecture | x86_64 |
| Platform | Ubuntu 20.04 |
| Processor | Core i7-8700k |
| RAM | 48GB DDR4 3200MHz |

The remaining steps (e.g., reading the image from local storage) are irrelevant in this context as explained in section 4.2.1. Addressing the second concern can be achieved by using static compilation. Static compilation ensures no external dependencies exist in the produced binary (the TEE process). External dependencies must be manually included in the binary, which we have made sure not to do.

## 4.5 Evaluation & Analysis

We have compared three main runs for testing: 1) full operation in REE, 2) Split-QR decoding with dynamic and static QR, 3) server decoding under dynamic and static QR. The idea is to test the secure payment method against unsecure REE method and compare with the previous design where decoding is done by the server. The full REE test defines the control data for best possible performance, given it does not use any TEE overhead. This reflects normal operation procedure that most applications have to use, given the lack of the TEE services.

The TEE testing is the true test in terms of security. Here we have installed only 12% of the ZBar library and 60% of the core decoder in the TrustZone as discussed in the design section. We ensured data integrity which is the main objective. Testing performance compared to the control

is important to ensure our solution design does not introduce major overheads, which significantly slow down normal operations. The server test is not fully representative, because it assumes all servers would operate with the same decoder we are using, and it assumes minimum server delays which can grow much longer in real-time settings. Either way, the test is mainly to measure the overhead reduction by the new design under ideal conditions compared to other settings. Given the server decoding approach relies more on server communication, delays introduced by connection issues would increase overhead even further and are not counted here.

### 4.5.1 Datasets

Typically, we would not use datasets for testing security applications. However, we need to measure performance over large variance of QR codes that come in different formats and sizes.

Table 4.5 Datasets descriptions [53]

| Dataset | Count | Resolution | Difficulty |
|---|---|---|---|
| Simple | 10,000 | 250x250 – 400x400 | Centered & aligned. |
| Complex | 1,602 | 100x100 – 2048x1536 | Random placement. Unaligned. Sheered. Skewed. |

Therefore, we have tested decoding on two sets. The first is simple, uniform, with easy to codes to decode. The set has 10k QR codes. Each image contains only the QR code, and it is perfectly squared. Furthermore, the image size varies between 250x250 to 400x400 pixels for different images. The second set is more realistic and has much more variance in shape and code format. It contains 1602 images with dimensions varying widely from low to high resolutions. The codes are also skewed and sheered in some images making it more difficult to decode. Table 4.5 shows the sets description.

### 4.5.2 Performance Comparison

Table 4.6 shows the processing time for the codes of the two sets run one by one, meaning call each code separately, which is the normal use. The complex set time consumption was 5 times longer than the simple ones on average, while some codes took 50 times longer. Furthermore, the table shows another test on both sets, where the images were taken in batches of 10 instead of singles. This test is supposed to measure the overhead introduced by initialization of the decoder library. For the simple codes set, processing time was reduced by half while the complex set showed around 12% decrease. In both cases, decoder initialization took nearly 10 milliseconds. Finally, Table 4.6 shows test results for singular and batched runs on both sets after resizing images to one size per set. We conclude from Table 4.6 that image size plays a crucial role in defining the processing time of decoding. The larger the image, the longer it takes, and the relation is nearly linear, and that is expected. We can also see the batched runs consistently saved ~12.5ms for all runs. This means the decoder initialization time is constant, independent of the image size. We have tested on many different image formats (PNG, JPG, BMP & RAW) and the results were pretty consistent.

Table 4.6 Results of eight calibration runs on the two datasets in the REE.

| Test run | Average time (ms) | Image dimensions |
|---|---|---|
| Simple – singular | 19.7 | Dataset original |
| Simple – batches of 10 | 08.2 | |
| Complex – singular | 97.8 | |
| Complex – batches of 10 | 84.0 | |
| Simple - singular | 21.3 | Resized: 400x400 |
| Simple – batches of 10 | 09.2 | |
| Complex – singular | 44.7 | Resized: 800x600 |
| Complex – batches of 10 | 32.5 | |

For the Split-QR method, we used the same singular runs of Table 4.6. The results are shown in Table 4.7. The overhead, as percentage, is largest for simple-singular images and least for complex-singular. This is quite expected given the Split-QR design adds near constant overhead to the procedure. In other words, the larger the image, the less overhead percentage is caused by Split-QR.

Table 4.7 Results of Split-QR runs.

| Test run | Average time (ms) | Overhead (ms) | Image dimensions |
|---|---|---|---|
| Simple – singular | 28.9 | 9.2 | Dataset original |
| Complex – singular | 109.2 | 11.4 | |
| Simple – singular | 30.9 | 9.6 | Fixed: 400x400 |
| Complex – singular | 55.4 | 10.7 | Fixed: 800x600 |

Finally, server decoding test was conducted similar to the Split-QR run, but the communication time is different. The results are shown in Table 4.8. It is important to note that Split-SSL overhead is not accounted for in these tests given we have simulated that part as explained in the design section. This will be discussed further shortly. In all cases we observe that there is similar overhead for small images, but the overhead grows massively for large sized and complex images. In the following we discuss the total overhead of the entire process, all the way from capturing the image to completion of the payment using the final attestation.

Table 4.8 Results for server decoder runs.

| Test run | Average time (ms) | Overhead (ms) | Image dimensions |
|---|---|---|---|
| Simple – singular | 32.7 | 13 | Dataset original |
| Complex – singular | 146.9 | 52.9 | |
| Simple – singular | 35.6 | 14.3 | Fixed: 400x400 |
| Complex – singular | 68.4 | 23.7 | Fixed: 800x600 |

### 4.5.3 Complete Analysis

Table 4.9 shows the full design time slices and compares different approaches in total time and overhead. Table 4.10 describes the different variables and parameters used in Table 4.9. As shown in Tables 4.8 and 4.9, the shortest total time is when running full REE. But it is also not secure and prone to attacks. The security provided by the other two methods is quite similar under the same assumptions. However, Split-QR is much faster with much smaller overhead especially when we factor in the overhead introduced by Split-SSL [47].

Table 4.9 Total processing time analysis for the different methods.

| Method | Read | Decoding | Amount | Server Comm. | Attestation | Total |
|---|---|---|---|---|---|---|
| REE | C | D | 0 | x | U | A + x |
| Split-QR | C | D + T | 0 | x + S | U | A + T + x + S |
| Split-QR static | C | D + T | K | x + S | U | A + T + K + x + S |
| Server normal | C | D | 0 | nx + S | U | A + nx + S |
| Server - static | C | D | K | nx + 2S | U | A + K + nx + 2S |

Furthermore, image transfer costs a lot of time depending on the network. In ideal test conditions, such times are minimal. In real-time scenarios, they are not predictable. Split-QR is not as affected on network transmission fluctuations, given it sends only one time, and the data is quite small. Split-QR does introduce T overhead, not present in other methods. However, as demonstrated in Table 4.10, T is quite small and does not grow fast with the image size.

Table 4.10 Descriptions of variables.

| Variable | Description | Typical value (ms) |
|---|---|---|
| C | Time required to capture an image from the camera device. | 10-20ms |
| D | ZBar decoding time. | +20ms – can grow to over 250ms for large images. |
| T | Overhead introduced by Split-QR on decoding. | +15ms – grows slowly with image size. |
| K | Overhead introduced by Keyboard TA from TruZ-Droid. | 160ms on ARM. In x86 environment it is projected to be around 40-50ms. |
| x | Transmission time to the server. | Depends on network. |
| nx | x multiplied by the number of transmissions necessary to send image data to the server. | Depends on network. |
| S | Overhead introduced by Split-SSL. | 370ms on ARM. In x86 environment it is projected to be around 80-90ms. |
| U | Overhead introduced by UI TA. | 60ms on ARM. In x86 environment it is projected to be around 15-20ms. |
| A | For convenience: A = C + D + U | - |

### 4.5.4 Summary of Results

Table 4.11 shows the summary of results produced in previous tests for all of the three methods.

Table 4.12 compares the methods in terms TEE size, feasibility, and integrity achievability.

Table 4.11 Methods performance: average decoding/code for code types and call settings (values in milliseconds)

| Method\Dataset | Simple codes 10k set batch | | Complex codes 1.6k set | |
|---|---|---|---|---|
| | singular | batches of 10 | singular | batch of 10 |
| RE decoding | 19.7 | 8.2 | 97.8 | 84.0 |
| Split TEE decoding | 28.9 | - | 109.2 | - |
| Server with same decoder | 32.7 | - | 146.9 | - |

Table 4.12 Operational parameters for the methods of decoding

| Test | Split | Full REE | Server/same decoder |
|---|---|---|---|
| TEE LoCs | 3700 + 500 | 0 | 500 |
| Feasibility | yes | yes | yes |
| Integrity | yes | no | yes |

## 4.6 Conclusions and Future Work

### 4.6.1 Conclusions

In this chapter, we introduced a new hybrid approach to solve the QR decoder problem. The design solution we introduced is novel and makes it much easier for developers to incorporate into existing applications compared to our previous approach when the server decoded images instead. The overall results show less overhead with increased transparency and seamless integration. Therefore, this new approach is quite advantageous. The simulation environment was quite sufficient to test the new design given it relies on existing independent components produced by TruZ-Droid and our previous work. This new decoder design can be extended beyond QR payments and towards anything that requires trusted QR code analysis where integrity of data is crucial. Secrecy would be harder to achieve in the given boundaries. The camera is available to REE at all times except when TEE needs it. Therefore, a QR code displayed on another screen will be hard to keep away from a compromised REE's reach. This is why our hybrid approach works well and makes a difference under the assumption that REE is compromised. We plan to extend this hybrid problem solving approach towards other services as shown in the following section. Table 4.13 summarizes the design highlights compared to other works and highlights the differences.

Table 4.13 Design highlights and comparison with other works summary

| Design | Highlights | Comparison to our design |
|---|---|---|
| Split-QR | Offers quick processing of QR codes that offers integrity assurances. Reduces QR payment processing overhead while maintaining a small TCB | N/A |
| Chapter 3 | QR payment using an external server for decoding. Offers same integrity assurances and a small TCB. Relies on remote server to process the QR code itself. | Smaller TCB size than Split-QR but larger performance overhead due to increased communication with the remote server. |
| Jeng-An Lin and Chiou-Shann Fuh [50] | Emphasis on optimizing decoding with focus on performance and accuracy. Uses Google open-source 1D/2D barcode image processing library Zxing-2.1. | Useful for optimizing decoding in the TrustZone option, but does not address decoding and data integrity through the TEE. |
| Nivedan Bhardwaj et. al. [51] | Android-based REE decoding system with colored barcode and measuring performance. | Optimizes decoding in the REE but does not address data integrity risks. |
| Madeline J Schrier [52] | Addresses the UPC barcode, which is widely used in retail and commerce. Optimizes parameter extraction from the features of the UPC symbology, which allows exact decoding. | Does not address QR codes nor splitting between TEE and REE, but can be very useful for secure UPC decoders implementations in the TEE. |

### 4.6.2 Future Work

Extending Split functions for trusted operations: The success of Split-SSL and the Split-QR decoding as outlined in this paper suggest extending the splitting to other applications and functionalities. Five types of different nature arise.

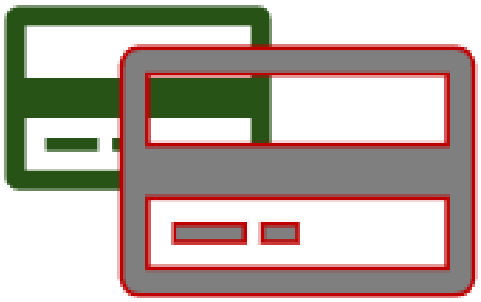
1. Small size services that fully need security—cannot split and must go to the TrustZone; Examples: GPS through a sensor. We estimate such services to be around 10% of services.
2. Services that can be split into essential to run on TEE, and not essential that is possible to run on REE with no harm to the basics of the operation or data integrity. They are the main candidates for generalization. Examples: QR decoding. We estimate such services to be around 60% of possible services.
3. Service that are possible to split, but the part for REE execution can harm the operation or beyond the REE capacity. This case can be due to complexity and a need of help from more powerful services. Examples: complex machine learning classification. We estimate such services to be around 10% of services.
4. Services that cannot be split and require outside dedicated services. Either fully TEE or fully external service. We estimate such services to be around 10% of services.
5. Large services that cannot be split and no external service are available. Only TEE or the REE can do them. Such cases are out of the scope definition assuming the Android is corrupted. Example: internet-assisted location attestation. We estimate such services to be around 10% of services.

Without splitting, all service types have to be done through external servers or others. After classification we can reduce the types that need full outside support to maybe 10%, and the split with external extensive use to 20% and around 70% can utilize local splitting or full TEE

execution. The splitting then serves 70% of the total load. The others can still benefit from the TrustZone albeit traditionally. In the upcoming work we will address these types and show the benefits to performance, integrity, and sometimes security.

# Chapter 5

# Dynamic Offline TrustZone VCC Transactions Generator



# Chapter 5: Dynamic Offline TrustZone VCC (DOT-VCC) Transactions Generator

**Abstract**. This chapter introduces DOT-VCC design which uniquely combines the benefits of VCCs and user-end tokenization systems and offers security for users against attacks targeting user devices as well as service providers. It works for in-person transactions (unlike regular VCCs) and does not require changes to infrastructure (unlike tokenization). Finally, it is compatible with merchant-end tokenization. The design offers protection for user cards information from theft or abuse in relation to merchant or service providers databases, and against attacks targeting user devices. The offline nature of the algorithm cuts off many attack surfaces. The DOT-VCC generator extends the utility of VCCs to cover in-person transactions unlike traditional online-only systems. The dynamic and offline aspects allow the generator to work independently from online communication to any remote servers. The design does not require architectural changes to the existing credit card network. The only modifications required are at the endpoints, the cardholder and bank server. The system can be utilized by banks and card issuers without specific settings and integrated within any TrustZone system as a service for any bank without contacting TrustZone specific manufacturers. The generated VCCs are irreversible which offers protection against database breaches, and the data is inaccessible for live attackers who compromise user devices. The issuing bank server and the TEE are the only parties who can reconstruct the number. The credit card number is based on secrets, original credit card data, and UTC timestamps and other dynamic parameters. The method uniquely combines various technologies that yield the unique security and services. Strong hash encryption is used and any change in the information blocks the transactions. The algorithm is robust even in the worst-case scenario where both the merchant database and the REE are compromised by attackers.

## 5.1 Introduction

Credit card information is one of the most valuable user data that attackers want to obtain. There are many types of attacks against many different environments. A very common attack targets merchant or service provider databases. Such merchants have massive databases of user data, and a security breach can leak financial information of customers including credit card information.

Virtual credit cards have existed for years, and many algorithms were developed to generate such cards. There are usually two main types of virtual credit cards: 1) merchant specific cards, and 2) limited use cards. The former type is generated specifically for each merchant and can be used many times. The latter type works with any destination, but it is limited in the number of times it can be used. In other words, it expires.

Virtual credit cards mitigate the breaches problem in merchant databases as users can cancel such cards immediately after usage. Furthermore, in case of a breach, the merchant can share the leaked credit cards with the respective issuers who can proceed to cancel such virtual cards with ease. Limited-use cards provide more protection due to expiry.

Most of the time, people use virtual credit cards for online purchases where they either use direct generation services or third-party generation services. Direct generators are provided by the same bank that issues the real credit card. Third-party generators are provided by other means. Either way, these cards are not commonly used for in-person payments. While some service providers have mobile phone applications to generate such data, it can be risky due to vulnerable operating systems. In a scenario where a remote attacker has full access over the operating system, the dynamically generated cards can be stolen the moment they reach the application.

Given virtual credit cards provide protection against merchant-based leaks, they do not provide solutions to attacks against the user requesting the cards which puts such cards at risk. On the

other hand, user-end protections often overlook virtual credit cards given they are mostly used for online payments. Instead, such protections focus on actual credit cards and protect them using trust-based environments (TrustZone and Apple's Secure Enclave). Unfortunately, those protections do not apply against attacks targeting merchants and service providers. Therefore, there are no solution designs that provide protection against attacks targeting both, merchants, and user devices.

While TrustZone supports different types of payments using the concept of Trusted Execution Environment (TEE), it does not extend the support for developers who wish to protect the data from leaking into the Rich Execution Environment (REE). In this case, generated virtual cards would be freely accessible to remote attackers which reduces the reaction time of users to stolen credit card information and causes more risks of theft.

In Chapter 3, we utilized the TrustZone to protect data integrity of peripheral devices (camera and location devices). Information from camera and location devices is often used in merchant-presented QR payments to attest and verify payment information. We ensured the data integrity all the way from the hardware device and to the server. TruZ-Droid [31] [32] was useful in achieving our goals due to Split-SSL which allows the TEE to prepare encrypted data to be sent to the server without the REE intercepting the data or altering it. In Chapter 4, we developed the Split-QR decoder by adding the essential parts of the decoder within the TrustZone. In a that work we have extensively tested such operation and shown the feasibility and analyzed the performance of the added protection. The designs from Chapters 3 and 4 address attacks that target merchant-presented QR transactions.

In this chapter, we introduce a novel design to enhance the security of credit card use. We target the problem of virtual credit card generation and consider the potential risks of a compromised

REE on the generated data. A trivial approach may attempt solving the problem by generating the data on the server and using Split-SSL to transfer the data to the TEE where it can be protected. However, such approach could introduce new issues in the TEE given it receives data from remote servers. A user mistake may authorize an untrustworthy server to store data in the TEE. Our approach to this problem is to cut off communication with remote servers while also completely masking users' credit card information. Such an approach demands offline synchronization between the user's device and the bank's server but without direct communication, and it works similar to authenticator applications. In other words, it can be fully synchronized even if the device is offline. The generated data is transferred over the credit card network, through the merchants and service providers, to the bank. The generated data must contain multiple pieces of information and can never be reversible. That is, the generated credit card information cannot be used to extract the original data. Once the merchant receives this payment information, it is sent to the credit cards network which forwards the data to the issuer where the payment can be processed. Given the synchronization, the issuer can verify the virtual card with ease. Section 5.2 covers background about credit card processing and virtual credit card generation. Section 5.3 defines the problem and what exactly are the terms and conditions of the REE where it is fully compromised. Section 5.4 outlines the solution design including all requirements for authentication by the bank, and algorithms needed for the number generation. Section 5.5 presents the implementation. Section 5.6 covers evaluation and testing, and section 5.7 outlines the main conclusions and future work.

## 5.2 Background and Related Work

Credit card use is expanding beyond expectation and grows well above linearly with the population growth. With that comes fraud and attempts by many parties to steal the cards data

and financial credit. In a comprehensive study [58] they predict the losses due to frauds in 2018-23 to get over $130 billion. This can impact trust badly and can destroy businesses too. The credit card trust problem is old as the cards have been. Traditionally, someone would steal cards from others and simply use them. They used to have signatures, but that can be faked with no expert to watch at the selling side. Then the idea came to run the service over the internet where card holders input a credit card verification (CCV) code and expiry date. There are several methods of defeating this security mechanism and it was estimated not to take longer than 6 seconds to brute force [11]. What is equally risky are many sellers themselves if they can charge the same card many times using the information provided by the original card transaction. With soft authentication comes easy breaches. Additional restrictions were applied, but the pace of corruption is generally faster.

A new novel idea came to issue one-time use credit cards instead of using the cards repeatedly. While this can work properly, it carries new associated risks. Usually, the cards are generated by the issuer and then transferred to the user over the internet. Any breach at any stage could steal the data and use it before the user gets a chance to use it. Furthermore, such approach is very difficult to utilize for in-person payment systems.

An early idea was proposed in 2001 by [66]. They have suggested an offline issuance of the number without requesting the bank-based issuance. The concept assigns the number by an independent entity that can serve the cardholder. We do not have the evolution history of such proposal, but if that service is done over insecure protocols the end result may fail just like the other ideas.

Finally came the TrustZone and our proposal to engage both sides. Restrict the offline issuance of the number to the TrustZone, which can protect the data fully by definition. In such

environment, REE corruption cannot steal anything from the TEE unless the user decides to write down the data on paper and then voluntarily give to the REE. That would be an effort from the cardholder to knowingly violate the secrecy of their own data. Even then, the damage is mitigated given the generation algorithm is time-bound and changes continuously.

### 5.2.1 Overview of Fraud Impact

The B2B (business-to-business) world ecommerce market is about $28 trillion in 2024 and the B2C (business-to-consumer) ecommerce market value is about $4.1 trillion. [74], [75], [76], [77]. Since 2020, ecommerce market has been on steady growth and the B2C market is expected to reach over $5.5 trillion by 2027. Fraudsters are tempted to taking advantage, and online payment fraud is rising fast. Two types of fraud, one is using the physical card and the other when card is not present, and its information is used over the phone or online. They are prime targets as they only need the card details stored digitally, and it is easier to get away with it [58].

Payment fraud affected 82% of organizations in 2018. Online sellers are estimated to lose $130 billion to online fraud between 2018 and 2023 MRC survey results show world-wide fraud costs 1.8% of business revenue. For every $1 of fraud from chargebacks, businesses lose an extra $2.94 which include fees, merchandise distribution, fraud investigation, legal prosecution, and software security. For customers, victims spend two working days cancelling their cards and dealing with the aftermath. For online sellers, fraud is a huge cost and the top concern for 44% of finance professionals. In Europe, the revised Payment Services Directive (PSD) means sellers will be legally responsible for fraud across their entire portfolio of online sellers [58].

There are different types of online payment fraud. One is when real customers deny receiving the goods, while they did, and they file a chargeback through their bank instead of requesting a refund from the seller. Most online payment fraud is identity theft:  Criminals steal cardholder

information through skimming on payment pages or buy on the dark web. In stolen cards, fraudsters impersonate the cardholder and buy online. The seller thinks the purchase is valid and sends the goods to the fraudster. In chargeback fraud, the cardholder sees the charges and contacts the bank. The seller is hit with a chargeback plus fees.

For the average fraudster, buying card details on the dark web is the easiest to get large numbers of card details. The Breach Level Index reports that over 14 billion data records have been stolen and leaked online since 2013 [73]. Fraudsters are stealthy, constantly finding new ways to improving their techniques. The dark web is a corner of the internet where criminals can interact with little trace. There are card details from all over the world on the dark web. Fraudsters buy compromised cards details and can quickly find out who the cards are registered to, and then spoof the location.

Fraudsters can buy real customer phone numbers online with card details - but they won't have access to actual owners' phones. To get around this they contact the phone company to request diverting the calls to their own numbers so that they can verify purchases if needed. The dark web also advertises 'calling services' where someone can call a victim's bank and credit card provider to change their registered phone number. Visa and Master card lost $750 million to credit card fraud for the period 1988-1998. That led to the creation of monitoring programs for chargebacks [58].

### 5.2.2 Credit Cards Concepts

**Credit cards coding.** Credit cards have different structures and standards. Here we outline the ISO/IEC 7812-1:2006 standard for 16-digit card numbers. The breakdown is as follows [59].

1. Digits 1-6: Issue Identifier Number (IIN). The first digit in this segment represents the issuer industry (e.g., 4 for Visa, 5 for MasterCard). The six digits provide a unique identifier for the

issuer institution. This is crucial in order for the credit cards network to find the proper destination when a payment is made.

2. Digits 7-15: Unique Personal Identifier (UPI). These digits represent the unique identity of the cardholder. They vary depending on the issuer and the generation method.
3. Digit 16: Luhn digit used to verify the validity of the credit card number. It is a checksum which adds to modulus 10 if the card number is properly constructed.

**The Luhn algorithm.** The "modulus 10" algorithm, is a checksum used to validate a variety of identification numbers, such as credit card, IMEI, and National Provider Identifier numbers in the US, and similar numbers in other countries. The Luhn formula [60] was created in the late 1950s by a group of mathematicians, and soon afterward, credit card companies adopted it. Most credit cards and many government identification numbers use the algorithm as a simple method of distinguishing valid numbers. It helps protect against accidental errors, not malicious attacks. Steps involved in the Luhn algorithm:

1. Starting from the rightmost digit, double the value of every second digit.
2. If doubling of a number results in a 2-digit number (e.g., $6 \times 2 = 12$), then add the digits of the product (e.g., 12: $1 + 2 = 3$), to get a single digit number.
3. Take the sum of all the digits.
4. If the total modulo 10 is equal to 0 then the number is valid; else, it is not valid.

**Credit card verification number generation**. Credit cards often carry a three-digit number at the back. In American Express cards, the number consists of four digits instead and it is calculated quite differently. Table 5.1 shows a three-digit number CCV generation.

**Virtual card numbers (VCC/VC).** These numbers provide seclusion, without compromising making payments or earning rewards. They improve making payments. They can be issued immediately and revoked or customized after use. Fraud is reduced with transaction-specific using VC. For commercial card programs, costs are also reduced. VC are very similar to physical credit or charge cards. They have a 16-digits, generated, and assigned for use by a user, and VC can be used immediately, if funding is available. They also provide spending controls, reporting tools and fraud protections and prevent misuse. For consumers, VC provide privacy protection against tracking by adversaries, while protecting sensitive banking information, including the real debit, credit card account number, expiration date and security code. For businesses, VC provides immediate issuance and funds, while managing cash flow, enhanced payables, and stronger reporting. For consumers, they can be generated with existing credit card accounts [62].

Table 5.1 Credit card verification (CCV) number generation algorithm [61].

| |
|---|
| 1. Generate a 16-byte key (32 hex digits). |
| 2. Retrieve credit card number (16 or 19 digits) which is the PAN. |
| 3. Retrieve 4 characters of the expiration date (formatted as YYmm). |
| 4. Use the proper three-digit service code (depends on industry and purpose, iCVC uses 999). |
| 5. Concatenate PAN, expiration date, and service code in that order to a single string. |
| 6. Pad the result from (5) with zeros on the right until reaching 32 in length. Call this data. |
| 7. DES encrypt the first half of the data with the first half of the key. |
| 8. XOR the result from (7) with the second half of data. |
| 9. DES encrypt the result from (8) with the first half of the key. |
| 10. Decrypt the result from (9) with the second half of the key. |
| 11. Encrypt the result from (10) with the first half of the key. |
| 12. Get the first digits (3 or 4) from the result as the calculated code. |

Single or multi-use VC can prevent fraud. Most credit card companies provide examples of both. Single-use works for one transaction, while multi-use can be used as a dedicated payment

method to a regular vendor, enhancing tracking and privacy. VC numbers differ from payment apps used by providers where most merchants can accept online purchases or in-store use with NFC (Near Field Communications) or MST (magnetic secure transmission) [83]. With these payment apps, a payment token replaces the primary account number (PAN). VC numbers can make purchases with online payments. These VC numbers are managed by the card issuer, where one can create VC numbers with any expiration date or spending limit suitable for the user with few extra steps to generate a VC number.

Returning items ordered online with VCC to a physical location could be a challenge. Some stores require inserting the card used for the purchase to refund back. The time it takes for a refund to show up on the credit card varies by vendors and card issuers. This is not possible with a VCC. Recent developments have provided solution for refund of VCC transactions. VCC with short expiration dates can cause issues for subscriptions. To keep subscription active, an update of VCC number each time it expires is needed [63].

With the merchant having the billing address, card number, expiration date, and security code, some can abuse the customer card. The cardholder is not liable for unauthorized charges, but until reversing the transaction the account may not have credit for valid purchases. With VC numbers, this risk is eliminated [64].

**Tokenization**. The token is another method that can be used for purchases [65], [69]. It is different from VC numbers, but both keep sensitive card number information from reuse for fraudulent purposes. Reporting and spending controls are available for both tokenization and VC number payments, but the features differ between VC issuers and smartphone payment app. In our method we used the standard virtual number utilization dynamically which provide security superior to standard credit cards and EMV tokenization.

### 5.2.3 Related Work

**Aviel D. Rubin, Rebecca N. Wright [66]** proposed a system that reduces the risk of misuse of a card number without the need of secure contact and authentication with the card issuer before each transaction. The protocol generates tokens as for conventional account numbers and observe transaction restrictions before approval. The account number is the shared secret between the card issuer and the holder. The tokens, length and format are identical to the account number, allowing easy layering of the protocol on existing commerce infrastructures. The account number is converted into a symmetric cryptographic key, by using a hash function. The transaction rules are encoded using the symmetric key that may be utilized in the transaction and verified by a card issuer who receives the request information from the merchant. The card issuer decrypts the code using the symmetric key, verifies the information and approves the transaction. The functionality is controlled by the card holder. Still the operation needs the bank approval of the method. The main focus of the work is regarding online transactions, and it does not provide effective ways to deploy the solution into everyday transactions. Furthermore, the work does not target vulnerable generation environments which gain instant access over generated numbers.

**Ian Molloy et. al. [67]** propose a dynamic virtual credit card number scheme that protects against cards abuse. The VCC numbers are generated based on the real account number, billing information, secret key, and other information in coordination with the issuer. The secret password must be given to the issuer prior to any payment, the remaining information is extracted from the payment information provided by the merchant. While there are similarities with the solution we offer, it does not consider any timestamps, and it requires the user to manually specify expiry date, merchant information, transaction amount and the secret password. This process must be done all over for every transaction and it requires internet connectivity. Furthermore, the solution does not specify the means of deployment into an actual system, nor

does it utilize the concept of a TEE to protect the generation process. From a mathematical point-of-view, the solution is reliable. From a practical point-of-view, it can be very inconvenient especially in physical transactions. And given the lack of timestamps, the attack window can be quite wide for an attacker with remote access over the generator device. Our approach eliminates such risk by allowing a dynamic specification of a timestamp while also integrating and verifying the solution's validity in the TEE.

**Park Chan-ho and Park Chang seop [68]** propose a virtual credit card number payment scheme based on public key system for efficient authentication in card present transaction. They claim their scheme can authenticate efficiently in card present transaction by preregistering virtual credit card number based on cardholder's public key without PKI. There are common features with our method but less dynamic. Again, this solution does not address the problem of a vulnerable device used for the generation. Table shows DT-VCC and related work

### 5.2.4 The Preferred Embodiment

In the following discussion, we showcase the attack scenario on merchant databases, and discuss the existing countermeasures and their underlying limitations; namely, tokenization (user-end and merchant-end) [69] and virtual credit cards.

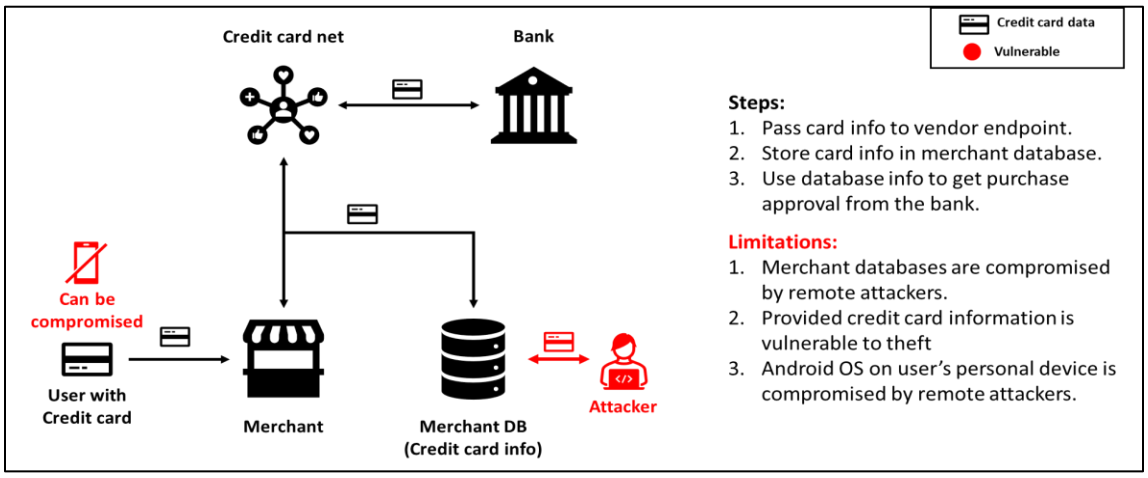



Figure 5.1 Baseline threat model attack.

One problem is their inability to provide a solution that covers both in-person and online transactions. We then compare the different methods to our implementation and summarize the differences. Figure 5.1 shows the general threat model attack against the merchant databases.

**A. User Card and Merchant Tokenization**

The most common method of protection for credit cards information is the use of tokenization. Tokenization is a general term that is often used for many different types of technologies used within the context of credit cards. Some of them are done on the user end while others are on the merchant's server.

**A1. User-end tokenization**

Figure 5.2 shows user-end tokenization countermeasure and limitations. The idea behind this form of tokenization is that you don't trust any merchant with sensitive information at any given stage.

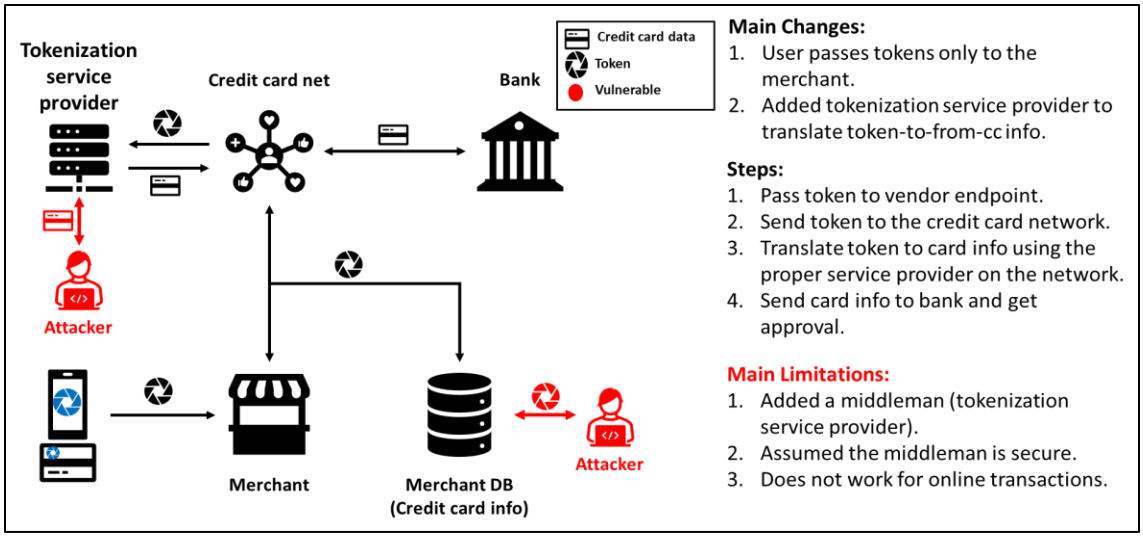



Figure 5.2 user-end tokenization countermeasure and limitations.

Therefore, the payment system only passes a token, which is a reference to an actual credit card, to a payment terminal (merchant), and also passes another token representing a single transaction. Those tokens are what merchant databases store, and accordingly, a compromised

database does not leak useful data. Such tokens need verification if the payment is to go through, and this is where the credit card network infrastructure comes into play. As stated before, the token is a reference to an actual account, this reference is stored in the service provider for the tokenization in the credit card network which has to translate it into the actual account [69]. Threat model vulnerability and counter measures limitation in comparisons with our solution that removes all threats.

The service provider for the tokenization comes in different types. Google Pay and Apple Pay use tokenization and they use their own services to store such references. Essentially, when you enter your card information, they will store them in their own databases, and give the mobile application a token reference to those accounts. Effectively, you trade an untrusted merchant with a "trusted" service provider. That is, you don't want a merchant to save your information, so you let Google or Apple save it for you and trust them to not leak your data. The same principle applies to EMV tokenization which is done on the credit card itself. While neither Google nor Apple will be responsible for the translation of your token, other similar service providers will be. In essence, the security concept is the same. This type of tokenization is used to protect in-person transactions where you directly pay the terminal.

**A2. Server or merchant-end tokenization:**

Figure 5.3 [70] shows the merchant tokenization countermeasure and limitations. In this type of tokenization, the merchant takes responsibility into not storing your account information. They will receive your sensitive information as it is, then tokenize it using a service provider. And the tokens also are references to the actual accounts. The service provider is the one that will hold your account information 'securely' while the merchant holds the token. Therefore, when you need to make a payment, the merchant will use the token with the service provider and the payment can then be processed and it goes through. Effectively, you are trusting the merchant to

properly protect your information through using an external service provider, that you are unaware of, to store your data and provide tokens to the merchant.

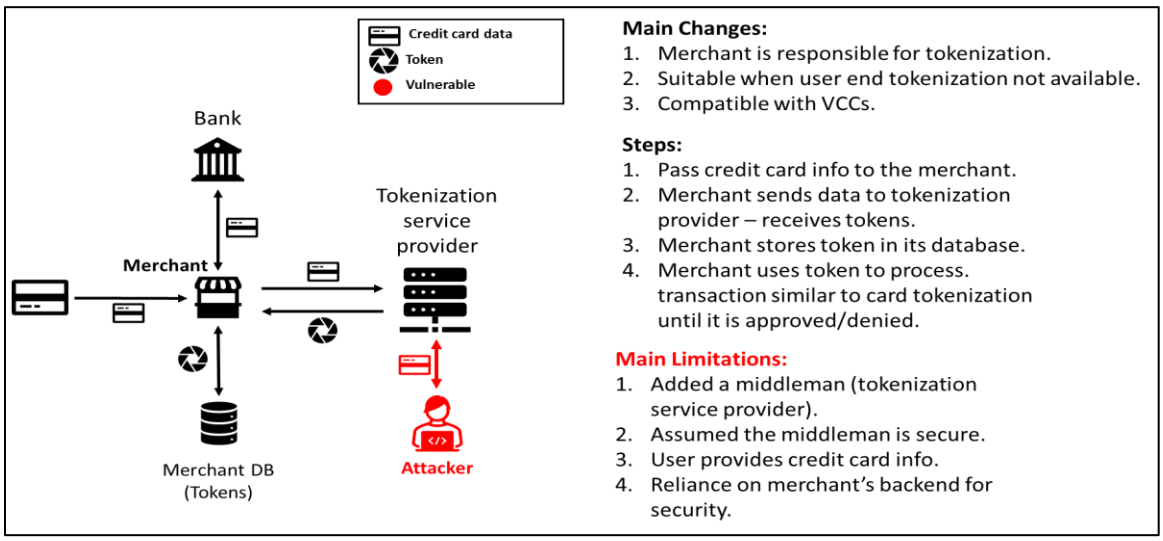



Figure 5.3 Merchant tokenization countermeasure and limitations.

It seems redundant to use this sort of tokenization when end-user tokenization already "solves" the problem. However, it is still relevant in ecommerce environments where the user has no means of transferring the token to the merchant. Therefore, the user relies on the merchant to be responsible and to tokenize their information for its protection.

**B. Virtual Credit Cards with Tokenization [71], [72]**

This type of protection is often used for ecommerce transactions where you do not want to trust the server to do the tokenization because not every server does it. In essence, VCCs are similar to user-end tokenization, except instead of tokens, we use virtually generated credit card numbers. However, it is highly different than the user-end tokenization, VCCs do not need any special infrastructure in the credit card network to work properly. The bank that generates the VCC can deal with it like a normal credit card because the bank's database knows how to distinguish or associate the two. Figure 5.4: shows the VCC tokenization countermeasures and limitations. VCCs are user-end protection methods which are supported by the servers of the users' banks.

That is a bit of an oversimplification since VCCs can also be generated by third-party providers, but the concept is the same.

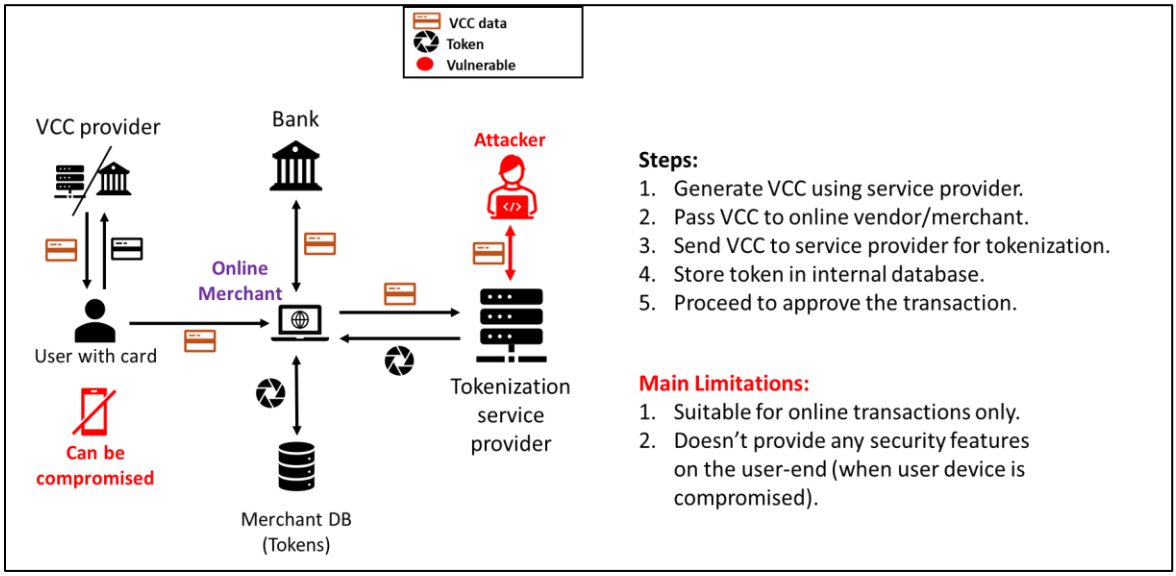



Figure 5.4 VCC with tokenization countermeasure and limitations.

VCCs are compatible with existing server-end tokenization systems, but do not rely on them in any way. That is, if the merchant's server implements tokenization, the transaction will still go through like a normal credit card transaction. The same can be said if there is a lack of tokenization by the merchant. The main issue with VCCs is their disconnect from in-person transactions. So, the only available technology for such transactions is going to be user-end tokenization, which requires massive infrastructure changes. And they still require the user to trust entities other than their bank to hold their sensitive data and not leak it, which leads into a transparency issue in cases of leakage.

## 5.3 Problem and Objectives

The main objective is to eliminate any damage caused by data breaches against merchant databases while providing transparent user experience. Mobile phones are perfect candidates for generating one-time use credit card numbers. However, they carry their own risks. Therefore, we put the following assumptions, based on the threat model in Figure 5.1, to define the scope:

1. Merchant databases are compromised: a seller who scans the user data is breached and an attacker has access to users' credit card data from the database.
2. Android OS on the mobile phone is compromised: an attacker has full root access over the operating system and can perform memory manipulation and management to steal any data that goes in and out of the environment.

We set the following criteria for the solution design:

a. The generated credit cards must be linked to real physical cards issued by valid authority, e.g., CapitalOne, Bank of America, Chase…etc.
b. Robust and secure credit card number generator. The user shall get approval from the credit card issuer to generate their own temporary numbers using a special key. The generated number shall be verifiable but irreversible. In other words, the data that goes into the generator cannot be derived from the output.
c. The integrity of the generated card must remain intact. The credit card issuer shall be able to verify the generated number immediately. Moreover, it must detect any data manipulation in any form or shape, whether by accident or malicious intent, during data transfer and examination at the issuer's server.
d. Credit card theft from merchant databases shall not put any risk on the user's data. This is achieved by the single-use aspect of the generated credit card number.
e. The generator must have the capacity to generate offline.
f. The design shall not require changes to the credit card network infrastructure, and hence provide easy adaptation by credit card issuers and banks.
g. User experience must remain simple and straightforward without inconvenience.

Finally, denial-of-service attacks are out of scope of this work as they are irrelevant to the problem being solved. Section 4 elaborates more on this scenario.

## 5.4 Solution Design (DOT-VCC Generator)

As stated earlier, mobile phones are good candidates to solve our problem. The vulnerable nature of the REE (Android or iOS operating systems) makes this problem more interesting. Therefore, we decided to utilize the concept of the ARM TrustZone and the TEE to secure the generation procedure and lock REE out of the operation. First, we describe the generation and then we showcase how the issuer can verify the correctness of the information.

This approach combines the best of both worlds, the VCCs and the end-user tokenization. It works for both ecommerce and in-person transactions, and it does not need any changes to the infrastructure of the credit card network. Figure 5.5 shows the DOT-VCC solution to eliminate dependence on merchants/providers for user security.

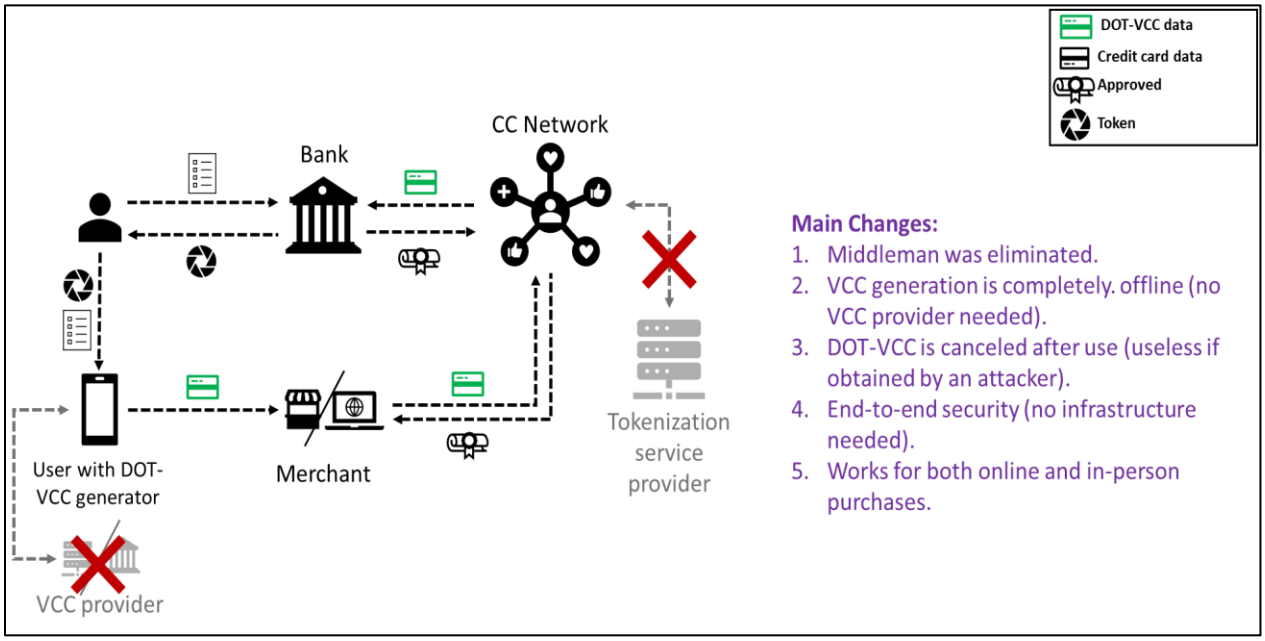



Figure 5.5: DOT-VCC design and the elimination of dependence on providers for security.

The main idea behind tokenization is trading one untrustworthy entity (merchant) with trustworthy service providers (e.g., Apple Pay or Google Pay) may be warranted due to them

exercising better security practices to protect user's data. Despite that being relatively accurate, it does not necessitate that such service provider is invulnerable.

In case a service provider is breached, more user data is put at risk due to having more data to be trusted with a service provider. By not relying on the service providers for security, we can eliminate a large attack surface that is completely out of the user's control. Figure 5.6 shows the steps taken in DOT-VCC design.

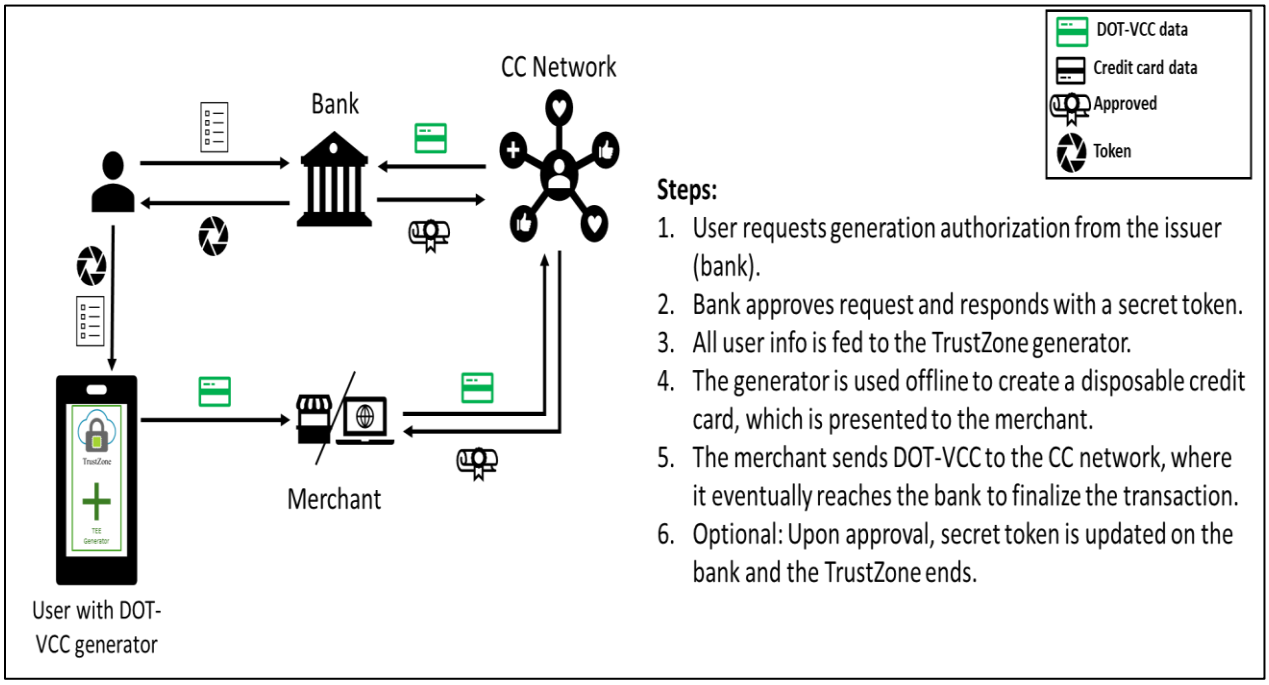



Figure 5.6: DOT-VCC design solution and steps.

Table 5.2 shows a comparison between the 4 methods including our solution. The main assumption in security is that nothing is secure for granted. "trusted" entity (service provider). Eventually, from an end-user point of view, a service provider as a role player is equal to the merchant within the token model.

Table 5.2 Comparison Table for the three types of services vs. our DOT-VCC Solution

| | User-end tokenization | Merchant-end tokenization | Virtual credit cards | Offline TrustZone VCC generator |
|---|---|---|---|---|
| Merchant gets real account info | No | Yes | No | No |
| Infrastructure change | Yes | Yes | No | No |
| Trust network | Issuing bank, Tokenization service provider | Issuing bank, merchant, Tokenization service provider and their databases | Issuing bank or VCC issuer | Issuing bank |
| End-to-end trust | No | No | Yes | Yes |
| Ecommerce | No | Yes | Yes | Yes |
| In-person | Yes | No | No | Yes |
| Mobile friendly | Yes | N/A | Yes | Yes |
| One-time use | Yes | Yes | Yes | Yes |
| Protection against user-end attacks | No | No | No | Yes, by design. |
| Delayed transaction processing | Yes | Yes | Yes | Optional |
| Secret code assignment | NA | NA | NA | Yes, changeable |
| QR codes | No | No | No | Yes |
| Dynamic offline | No | No | No | Yes |

### 5.4.1 Acquiring Issuer Approval

This is the initialization step. The user must have an existing credit card with the issuer. The issuer can be a bank or an authorized credit card company. For the majority of cases, it is usually a bank. Anyway, once the user has a credit card, they apply for an authorization to generate single-use credit cards that are linked to their own existing account. This can be a service sold by the issuer or provided for free. Either way, the issuer must generate a secret key that is only shared with the corresponding user. This key is needed to generate single-use credit card numbers. Once the key is given, the following data is provided to the credit card generator:

1. Original credit card number (16 digits).
2. Original credit card verification number (CVV, CVC, CCV, …, etc.).
3. Original credit card expiry date.
4. Cardholder name: first, middle initial (if any) and last.
5. The issuer-generated secret token.

The given information is stored securely in the TEE and REE is never allowed access to such data. The generator uses this data to instantly generate a single-time credit card number which is provided to the issuer during the transaction to verify correctness. After this, the approval step is processed. It is important to note that the issuer does not share the secret token it uses to generate the original credit card's verification code (CCV, CVV…). The secret token shared with the user is different and used only for the future single-use cards.

### 5.4.2 Constructing the Credit Card Number

In order to properly generate variable credit card information, we need to include in the generation a variable that keeps changing. We opted to use the Coordinated Universal Time (UTC) to get timestamps. However, we need to make sure the bank has the same timestamp at the time it receives the generated card information in order to verify properly. Therefore, we

decided to discretize timestamps. That is, we set duration in minutes which defines the time periods the timestamp changes. For example, if the amount is 5 minutes, the timestamp updates every five minutes and the time period 12:00:00-12:04:59 is represented by one timestamp. The following 5 minutes are defined by a new timestamp and so on. The construction string is shown in figure 5.7:

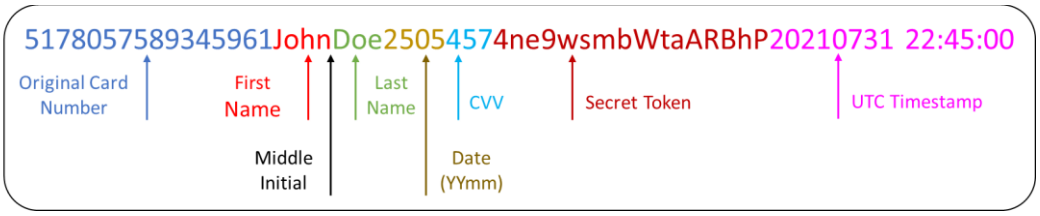



Figure 5.7 The concatenation string.

1. Retrieve the original card data and cardholder information from a secure storage.
2. Retrieve the current UTC timestamp.
3. Concatenate the information into an ASCII (8-bit) character string. The concatenation structure is shown in Figure 5.7.
4. Use the resulting concatenated string from step (3) as input to SHA256 hash function and produce a 32-byte hash.
5. Copy the first six digits from the original card as the first six digits of the new card. This helps the credit card network find the issuer when it needs to get authorization for the transaction.
6. Use the hash generated in step (4) to generate digits 7-15 in the new credit card number. One way to do that is to by using three bytes from the hash for every digit. For example, the leftmost bytes 1-3 represent digit 7 in the new card number, bytes 4-7 represent digit 8 and so on. In total, we use 27 bytes (9 digits x 3 bytes per digit). The last 5 bytes of the hash are not used. Now, the 3 bytes for each digit are XORed to produce 1 byte. This is then divided by 10 and the remainder is the final digit used in the credit card.

7. At this point, we have generated digits 1-15 of the new card number. We calculate the Luhn checksum, according to the algorithm described in section 2.2. With the checksum, the 16-digit card number is complete.
8. Generate expiry date for the new card. This step can be simple, take the current date and increment by a predefined number of years (e.g., 5 years).
9. Finally, we calculate the verification code as described in Table 5.1. For the secret token, we use the key provided by the issuer upon approval of the generation procedure.

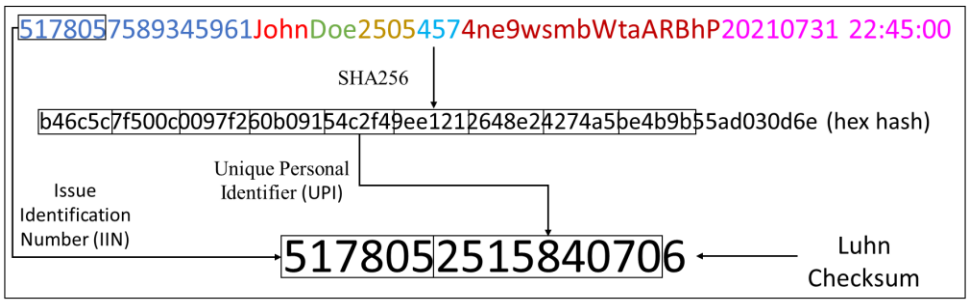



Figure 5.8 Credit card number construction.

Figure 5.8 shows a summary of the algorithm that generates the 16-digit card number. The figure shows the SHA256 hash in hexadecimal format (64 in total, each two hex digits represent one byte) because the ASCII representation does not have displayable text characters for all values. The secret token as shown in Figures 5.7 and 5.8 is for demonstration purpose. Actual keys can be longer and more complex, and infeasible to break by brute force or other methods when they are generated using proper randomizers.

### 5.4.3 Data Reconstruction by the Issuer (Bank)

Once the single-use credit card is generated, it is ready to use for payment. Given the data is inside the TEE, it can be transferred into the merchant's terminal in different ways. One method is to use Near Field Communication (NFC) similar to how Google Pay and Apple Pay do. However, unlike Google or Apple Pay apps, this will not be tokenized, and the information is

given directly to the payment terminal where it can be encrypted by the terminal prior to going into the credit card network. This is how swipe-payment procedures work as the data is encrypted after inception either using end-to-end (E2EE) or point-to-point (P2PE) encryption.

Either way, after the data is accepted by the terminal, the credit card network will be able to find the issuer industry (e.g., MasterCard, Visa, …, etc.) which will redirect the request to the corresponding issuer using the first six digits of the card number. Given we reused the same six digits as the original card, this should lead to the same issuer. The issuer verifies the received information as shown in Figure 5.9 and described below:

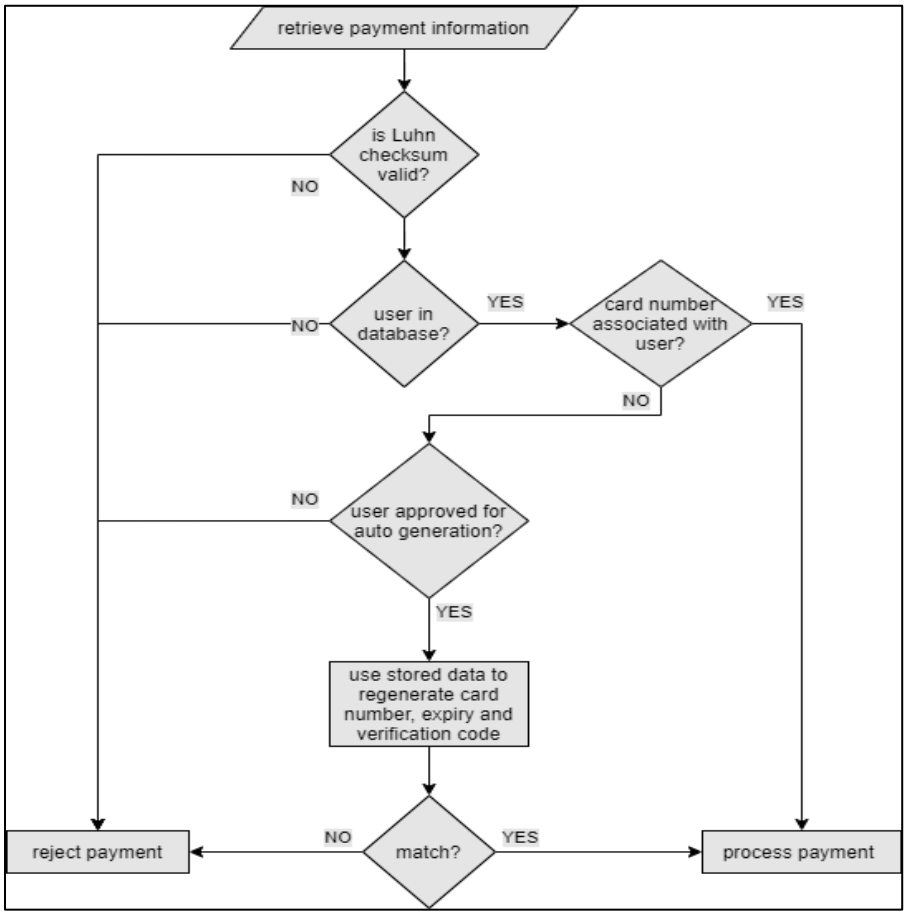



Figure 5.9 Flowchart of the overall verification process of generated credit card data.

1. Check Luhn checksum. Incorrect checksum means invalid card number which leads to the rejection of the payment on the spot.

2. Query the database for cardholder information, if not found in the database the payment is instantly rejected.

3. Query the database for existing cardholder credit cards. If there exists a card that matches the received number, then it is a real card issued directly by the bank and payment is further processed like normal payments.

4. If the query from step (3) returned nothing, the server checks if the cardholder is approved to generate their own single-use credit cards. If not, the payment is rejected.

5. Once the cardholder is known to be approved to generate cards, the server sends query to the database to get the approval using the associated secret token and original card information (including expiry and verification code). The server then generates the virtual card number from the data using the same procedure described in section 4.2.

6. The server-generated and the received numbers are compared. If they match the payment is processed and the generated data is stored and marked as used. If the same data is received again by the server, it will be rejected because it is marked as being used.

### 5.4.4 Other Design Considerations

Now that we covered the overall approval procedure, generation and verification of generated data, there are still some fine issues that can be considered. They are not critical, but we discuss them for completeness.

**Multiple real credit cards for a single cardholder**. It is possible for a cardholder to have multiple credit cards from the same issuer or bank. In such case, the procedure only considers the ones that have been approved for generation of single-time cards. That is, if the cardholder is approved for one but not the other, the secret token cannot be used by the user to generate for the

unapproved one. This association is made and stored by the issuer. The association is made between certain secret tokens with certain cards and never with the cardholder's profile. To avoid using a single key to generate multiple cards, the user is required to get a separate approval for each card. Other protocols can be envisioned.

**Periodic time-stamp overlap.** It is possible for the client to generate a card number near the end of a periodic timestamp. For example, generating the number at 12:04:59 (assuming 5-minute periods). In such case the data is more likely to be received by the issuer in the next timestamp period. The issuer then may use a different timestamp for the verification which leads to rejection of valid data. To accommodate such possibility, the issuer can check the previous periodic timestamp if the current live timestamp is within a few seconds from the end of the previous periodic timestamp. Alternatively, the bank may choose a policy to accept credit cards from the previous timestamp period. The time window would remain small in the context of data breach attacks.

**Collision between cards.** This problem does not exist within DOT-VCC**.** The PIN is 9 digit long and can cover up to one billion credit cards per IIN. While the range is wide, if such a design is utilized on scale then it is possible for the regular generated VCCs to collide with each other or with real credit cards. This is where the personal identification information come in handy given they are used in conjunction with the credit card number (whether real or VCC) to identify the specific card in question. In other words, the personal information will have to be the same with another random credit card holder for the two cards to be completely indistinguishable. The personal information contains name and address (which can be masked as a token for privacy concerns). The two users will have to have the same name and address for a true collision to happen. There is only one case where that may be true, when the user is requesting the DOT-

VCC when they generate their real credit card as a VCC. The odds of that happening are one in a billion, which is negligible compared to regulars credit cards. As a remedy, the generator compares the generated number with the real number of the user, and if they do match, the generation is skipped and another number is generated using the next/previous time periodic timestamp to avoid the collision. In that sense collision is a nonexistence problem for DOT-VCC, while it can happen with regular VCC except if the generators consider the remedy above. However, it would be more complex, because regular VCCs generation is not made in comparison with a certain true number. Compared to regular credit cards, regular VCC are still billion times better, while with DOT-VCC this is not possible.

**Multiple payments within the same timestamp**. It is possible for cardholders to initiate more than one transaction within a very short time period. This can be shorter than the associated timestamp period. Such operations can overlap with unauthorized operations. The issuer would have already marked this card as used and rejects further payments. A good remedy is to use a counter for the number of generations during the same timestamp and include it in the generation and verification. Such addition can completely eliminate any possible breach, and at the same time allow for multiple transactions during the same time stamp. We have not implemented such option. Another remedy can do by borrowing future timestamp. In this scenario, the generator and issuer keep track of the used generated cards. The generator marks the card used when sending the request to the payment terminal. The issuer does that when the payment is processed. If another generation is needed during this period, the generator uses the next timestamp to generate a new card number. The issuer receives the new card number and checks against the old card (if the timestamp is the same as the old). Since it is the second request, the server uses the

timestamp for the new number. And approves if there is a match. Again, we have not implemented this option.

**Theft of real credit card information**. It is possible for attackers to steal the original credit card information due to human errors. A potential solution could be to completely disable transactions using the original credit card upon the approval of the virtual generation procedure. Other options can be envisioned.

## 5.5 Implementation

TrustZone developments can be a challenging task quite often due to multiple reasons. For one, there is a limited set of hardware development boards that completely support the latest OP-TEE standards for development. Further, the development of OP-TEE is completely done in C in similar style to embedded system development, except it lacks the library support. Due to shortage of libraries, certain tasks are not possible right away.

For our problem here, the algorithm we developed is self-contained. While it does rely on certain cryptographic operations, it has very little other dependencies. Luckily, given the TrustZone is often used for cryptography, it contains a wide range of algorithms including SHA256 hash function and DES encryption algorithm. Those two algorithms are the only cryptography methods used in our generation algorithm. There are few other implementation challenges. Retrieving the input from the user while operating in the TrustZone is a problem on its own. Luckily, this was resolved by the TruZ-Droid UI and Keyboard [2] which was verified to allow user input into the TEE. Another issue is obtaining UTC timestamp within the TEE. OP-TEE implementation does not currently support getting real-time clock epochs except from the REE which cannot be trusted for sensitive operations like ours. We solved this issue in Chapter 3 [7] where we established a secure channel between the TEE and location sensor (GPS). Location

sensors receive UTC time directly from satellites embedded in NMEA statements. We can securely obtain these statements and hence the UTC timestamp. However, location data from satellites can be lost in large buildings due to poor reception. Therefore, the location data is only used for calibration of the TEE clock which can operate independently thereafter. While the location timestamps may not be extremely precise, they are good enough. In addition, we already covered the overlap case of periodic timestamp in section 4.4. Meanwhile, in a commercial device the manufacturer can implement an accurate clock and therefore this will not be an issue.

Given the above challenges established solutions we do not need to complicate our proof-of-concept outlined in this work. Therefore, we tested our system using OP-TEE with Qemu emulator for ARMv8. We essentially simulate the established solutions to simplify the proof-of-concept and study the algorithm more accurately.

## 5.6 Evaluation & Analysis

Evaluation of this system can be broken down into three main areas: validation of the algorithm and its correctness under different and extreme test cases to also test the robustness of the provided solution. We also need to conduct performance analysis and measure any possible overhead introduced by the design. Finally, we need to evaluate the impact of additional services we embedded in the Trusted Execution Environment (TEE). Therefore, we define a set of test cases and then show our results and findings.

### 5.6.1 Validating the algorithm correctness.

The main test cases for the algorithm are the following:

1. Case1: Approved generator with correct data: the cardholder is approved to generate single-use cards, the provided data is correct for the generator, and the secret token is correct. This is the only test case that should successfully complete a transaction.

2. Case2: Approved generator with incorrect data: the cardholder is approved as in (1), the secret token is correct, but the part of rest of the data is corrupted due to uncounted for reasons. For example, hardware fault that changes one character in the name. This case should fail.
3. Case3: Malicious generator with incorrect data: unapproved malicious actor attempting to generate new cards with incorrect initial data (card information, name, etc.). The secret token is incorrect as well. This should fail always.
4. Case4: Malicious generator with correct data: unapproved malicious actor with the correct data (card info, name, etc.) but without the secret token. We discussed this scenario in section 4.4. That is, the attacker managed to get the cardholder information through external means. If the original card is restricted from initiating any transactions, the malicious actor will have to use this information to attempt generating their own virtual credit card numbers. The missing piece of the puzzle is the secret token. This case should fail always.

'Data' refers to the initial provided information to the issuer (i.e., name, card number, verification code, and expiry date). The first test case should be the only one to successfully complete a transaction. Test cases 2-4 are completely different in nature and cause but would be treated with rejection equally from the perspective of the issuer trying to validate the generated data. Namely, the received information is incorrect and does not match what it had generated on its end. Therefore, all of them are rejected. This is a good expectation.

Table 5.3 shows the results of the generated credit card information for each of the 4 test cases with the addition of the generated data at the issuer server side. The tests were taken within the same timestamp. We can observe that the Issue Identification Number (IIN) is the same for test cases 1, 2, and 4 while it is completely different for 3. This is due to the fact that the original

credit card number is unknown in case 3 and hence the attacker must rely on different means of guessing. In such a case, the generated card will not even reach the proper issuer server given it points to a different IIN. Therefore, it will be rejected altogether. For test case 2 we changed only 1 bit in the first name of the cardholder and the result is a completely different Unique Personal Identity (UPI). This small change is reflected in the result due to the nature of one-way hash functions (namely, SHA256) which are very sensitive to any modification. For test case 4, everything is known to the attacker except the secret token, and the result is a completely different UPI as in test case 2 which, again, is due to the SHA256 hash function.

Table 5.3 Generated credit card numbers for the four test cases described above, and the reference generator on the issuer server side.

| Test Case | Generated Card | Modified Data |
|---|---|---|
| Case1 | 5178052475745861 | None - fully accurate data. |
| Case2 | 5178053685172813 | 1 bit changed from 0 to 1 in the first name. |
| Case3 | 4991470714805908 | Everything other than the name. |
| Case4 | 5178054174819278 | Secret token is not given (guessed) |
| Issuer | **5178052475745861** | **None** |

Due to the extreme sensitivity of SHA256 to any data modification, there is no room for guesswork. The attacker either has all the information or no use. Although, the attacker in test case 4 can attempt to guess the secret token by brute force, the task is computationally impossible. Strong secret tokens can contain 95 different possibilities (if special symbols, numbers, lower/upper-case letters…etc. are included) for each character. Even a small 16-byte key would take over $8\times10^{16}$ years to guess.

### 5.6.2 Performance

It is important to measure the time it takes the algorithm to generate new cards in order to test the practicality of the algorithm in real-life use cases. Therefore, we measured different time

components as outlined in Table 5.4 As we can see from the table, it takes nearly 145 milliseconds for the REE (Android) to initiate and send a request to the TrustZone generate a new card. This operation is long period due to the nature client application (CA) and trusted application (TA) flow. The operation includes opening a session, requesting the data, and getting an acknowledgement of the generation, all of which require context switching through the Secure Monitor. Either way, this is still within reasonable delay as it usually takes the cardholder a few seconds (thousands of milliseconds) to get their card out of their wallets. In practice, it will take the user much longer time to get the phone and initiate the request than the request procedure itself. The generation time, once inside the TEE, only takes 2.19 milliseconds which is less than 2% of the total procedure on the mobile phone. The algorithm is lightweight and portable.

Table 5.4 Time measurement analysis

| Function | Time (ms) | Description |
|---|---|---|
| REE request for new card | 145 | Total time taken for the REE to initiate a request to the TEE to generate a new credit card. This includes creating TEE session and context switching through the SMC. |
| Card generation | 2.19 | Time taken to generate a new card within the TEE (after the REE invokes the TA). |
| Server card generation | 0.079 | Time taken by the server to generate the card to check against received data. |

Finally, the server-side analysis can be quite complicated. We only focused on the part where the server decides to generate the card number (after other ordinary test are verified) to compare with the card number it received from the credit card network. Since it is a server environment, processing speeds are much higher and hence the generation only took 79 microseconds. In practice, this “overhead” is negligible given the database queries consume a lot longer. All things

considered; the design does not introduce any significant overhead compared to normal credit card processing methods.

### 5.6.3 TEE Size

It is always important to study the effects of embedding new functions into the TrustZone and its TEE. The model of the TEE is to have a small TCB which can be verified and tested for security purposes. Table 5.5 breaks down the functions added into the TEE and their corresponding size in LoCs (Lines of Code).

Table 5.5 Description of functions added into the TEE.

| Function | Added LoCs | Description |
|---|---|---|
| Time parser | 110 | Convert time (epochs) into string format. |
| Card generator | 190 | Use stored data to generate a new card. |
| Trusted Application | 200 | OP-TEE standard procedure to add TAs into the TEE |

The total count of LoCs added to the TEE is ~500. However, the code relevant to the functionality we developed is only ~300. That is, the 200 lines added by the TA itself follow the OP-TEE standard structure for TAs which are largely redundant in most TAs.

The nature of the introduced code is also safe for use. There is no input/output data flowing between the REE and TEE, and the procedure is internal within the TEE which does not introduce any new attack surfaces against the TEE itself.

## 5.7 Conclusions

### 5.7.1 Summary

There are numerous countermeasures that attempt to solve the payment fraud problem. In modern times, everyone holds a smartphone with great processing abilities. Our design proved it

is possible to utilize those devices fully securely to generate single-use credit cards for in-person and online payments whenever necessary.

The offline nature of the algorithm cuts off many attack surfaces and windows which attackers exploit. Furthermore, the algorithm does not impose any architectural changes to the existing credit card network. The only modifications required by the algorithm are at the very endpoints of the network (the cardholder and issuer server) while everything in-between remains the same.

The strong security does not come with any significant overhead and works very efficiently. Furthermore, it does not introduce risks to the TrustZone environment. The algorithm proved robust even in the worst-case scenario where both the payment terminal and the REE (Android) are compromised by attackers. In addition, the algorithm is quite convenient for cardholders. The procedure is straightforward and fully safe.

The algorithm, as-is, can be utilized by numerous banks and card issuers without any specification towards any of them. This allows them to utilize the same TEE service without having to contact smartphone vendors to support their own specific services. This, in turn, saves a lot of development effort, time and resources for all parties.

We mentioned NFC as a way to transfer the generated data from the TEE to the payment terminal. In future work, we plan to introduce QR encoder into the TEE which generates a QR code representing the newly generated credit card information. This extension allows merchants to scan user payment which is a system widely used in some parts of the world like China.

The framework can still accommodate bank generated card numbers, but the main advantages will not be optimized. Still the TrustZone handling will make it better than ordinary enquiry by the users from the bank.

The solutions offered in this work can provide near complete protection with seamless activity. The design is flexible to further improvements for convenience of use.

The overall Threat Model consists of two combined adversary models: Remote attacker against user-end device from the compromised Operating System, or against merchant databases. The Dynamic Offline TrustZone VCC Generator targets both merchant and buyer presented QR payments in addition to general payment services. It provides the following protections.

1. Provides VCC numbers for payment processing as part of regular credit card transaction, and after scanning merchant or buyer-presented QR codes.
2. The generated VCCs are never available to OS and cannot be used for database attacks.

In comparison existing market protections against merchant databases include User-end tokenization which work only for in-person transactions, while merchant-end tokenization and regular virtual credit cards: work only for ecommerce transactions.

The threat model consists of two combined adversary models: Remote attacker against user-end device from a compromised Operating System, or against merchant databases. The Dynamic Offline TrustZone VCC Generator targets both merchant and buyer presented QR payments in addition to general payment services. It offers the following services and protections:

3. Provides VCC numbers for payment processing as part of regular credit card transactions, and after scanning merchant or buyer-presented QR codes.
4. The generated VCCs are never available to OS and cannot be used for database attacks.

In comparison existing market protections against merchant databases include User-end tokenization, which work only for in-person transactions, while merchant-end tokenization and regular virtual credit cards work only for ecommerce transactions.

### 5.7.2 The System Services and Protections

1. Provides security against attacks targeting mobile-based or other credit card transactions, while assuming compromised environments both on the mobile device and merchant databases. The design fully secures user payment information and eliminates the risk of leakage, either through attacks against the operating system or the merchant databases.
2. Provides security and data integrity regarding transactions using QR payments. It can be used to initiate VCC payments after scanning merchant or buyer presented QR code transactions.
3. This system uniquely combines the benefits of VCCs and user-end tokenization systems and other parameters. It works with in-person transactions (unlike VCCs), does not require changes to any infrastructure (unlike tokenization), and it is compatible with merchant-end tokenization.
4. The construction method of the credit card number is based on secrets, original credit card data, and UTC time. The method uniquely combines many existing technologies that yield the unique levels of security and services. Strong hash encryption is used and any change in the card information blocks its use in transactions. The system uses UTC as the dynamic component of the VCC variability. This allows transactions within the same time window if assigned for single or multiple transactions.
5. The generated VCCs are irreversible and hence inaccessible to the Android or other OS and cannot be utilized by database attackers. It is not possible to get real information from them. The issuing bank and the TrustZone are the only parties who can reconstruct the number. The unique offline VCC generator extends their use to in-person transactions unlike traditional online-only role. The dynamic and offline nature allows the generator to work completely independently from online communication to any remote servers.

6. The secret between the bank and TrustZone user is initially made directly between the bank and the user. From there on the secret and other information can be updated without the need of manual interference, while providing full security. Attempts of denial of service do not yield any benefits to attacks, and they can be solved within a short practical time.
7. The method is fully implementable as a general service by banks and TrustZone vendors and can work for various mobile or other devices operating systems.
8. The robustness and ease of implementation answer the important questions in the most direct way, without reducing versatility, i.e., tokenization uses random numbers to generate tokens, and the design does not add complexity beyond the service providers trust-based network.
9. The use of VCCs eliminates any need for architectural changes to the credit card network. They provide end-to-end security that is completely transparent to the user and does not require the trust of any entity between the cardholder and the issuing bank.
10. The system is fast, lightweight, and does not introduce overhead. It can handle complex interacting transactions acting across credit card and QR services with seamless ease.
11. The small size of the generator keeps TrustZone TCB small and does not compromise the security of the TrustZone.
12. The current implementation serves Android but can extend easily to other platforms.

# Chapter 6

# A Framework for TrustZone Encoding/Decoding for QR Buyer-Presented and VCC Offline Generated Transactions

# Chapter 6: Secure End-to-End Mobile Financial Transactions Framework (SEEM-FTF)

**Abstract.** In this chapter we present a design of a framework for TrustZone encoding/decoding for QR buyer-presented and DOT-VCC offline generated transactions. The framework securely works offline and can operate as a standalone TrustProvider, or regular TrustZone connected to Android or similar systems. Implementing either mode (standalone/TrustZone) can provide full protection. The framework also provides other functions like encoding/decoding, that can enable for peer to peer closed circuit communication In Chapters 3 and 4 we introduced solutions for merchant-presented QR payments using the TrustZone by securing the data path for transactions with the server, and providing the Split-QR decoder for processing the data in the TEE. TruZ-Droid addressed buyer-presented QR codes, where the code is generated once and stored in the TEE. In Chapter 5, we have developed DOT-VCC which is a design that eliminates dependence on service providers for users' security by generating virtual credit card numbers offline to protect users credit card information from data breaches and attacks against mobile devices. In this work we connect all components together to introduce our Secure End-to-End Mobile Financial Transactions Framework (SEEM-FTF), which offers security for merchant-presented and buyer-presented QR transactions and extends towards in-person transactions as well as online transactions. We also discuss a design for standalone device (TrustProvider) compatible with the TrustZone systems and dedicated towards providing secure transactions and can replace carrying physical credit cards. The SEEM-FTF implements QR encoding in the TEE, which is combined with the DOT-VCC design for secure and dynamic buyer-presented QR transactions that do not rely on providers for security while offering low overhead, transparent user experience, compatibility with existing infrastructure, and are easy to integrate for system developers. The components work well to support end-to-end transactions with the components

presented in Chapters 3 and 4 where the merchant can just be another user of the SEEM-FTF design.

## 6.1 Introduction

This chapter presents the fourth component of a system development that provides high security and data integrity for financial transactions using the ARM TrustZone for credit card presentation using QR encoding. In Chapter 3 we protected the integrity of merchant-presented QR transactions using the TrustZone by ensuring the data path from the moment a request is made all the way to the completion of the transaction was attested by the TrustZone. Chapter 4 offered a unique method for QR decoding in the TrustZone using a small codebase by splitting the decoding between the TEE and REE. Decoding was tested in different domains and the split-decoder proved to offer better security, performance, and reliability compared to the alternatives. In Chapter 5 we have developed the DOT-VCC design that eliminates dependence on service providers for users' security by generating virtual credit card numbers offline to protect users credit card information from data breaches and attacks against mobile devices. In this chapter we introduce a method for presenting payment information to sellers using QR encoding from within the TrustZone. We designed and tested a system that extends the TrustZone functionality to generate QR codes for buyer transactions by installing an encoder within the TrustZone and ensuring sensitive information remains protected. It was done with minimum addition of Lines of Code (LoCs). There are cases where Near-Field Communication (NFC) can be used to transfer payment data from the TEE to the payment terminal to ensure security [83]. We use QR code presentation for the sellers to scan and complete the transaction. Combined with DOT-VCC, this method ensures the users' credit card information is never put at risk while simultaneously offering a convenient payment system. Another added benefit is the introduction of middleman-

free peer-to-peer payment method that can be natively supported by mobile devices while offering protection against data breaches.

The key motivation for the system development is the constant threat against data integrity and security in credit card transactions that lead to billions in losses yearly [1], [4], [11], [12]. Such threats are devastating because they target **merchant** and **service-provider** databases which when breached leak countless numbers of users' credit cards. Furthermore, attackers also target mobile devices to steal such data directly from users using different malware that propagates and compromises operating systems like Android and iOS and causes data leaks and can grant attackers full root access over the operating system. Such access allows attackers to perform memory manipulation and management to steal any outgoing or incoming data.

Building a system that can address these problems and provide a higher security with a framework that integrates many functions is a needed development. The system we are proposing can be implemented by individuals with no need of manufactures restrictions or limitations. In that sense this work covers the encoding component and the preliminary design configuration of an integrated system that can provide higher security levels. In section 6.2 we cover a background and related work. It also covers related work and conclusions towards building an integrated system of the services. Section 6.3 presents the problem and objectives. Section 6.4 outlines our new design solution that generates buyer present QR code using direct or generated virtual credit cards. In section 6.5 we present the implementation of the selected QR encoder that satisfies the TrustZone settings mainly using C language and has minimum size. The implementation connects all functionalities to produce an integrated system that serves a wide range of applications. In addition, it covers testing and evaluation of the implemented

solutions using an alternative of dedicated standalone trust provider. Finally, section 6.6 outlines conclusions and an outline of an integrated system core functionalities.

## 6.2 Background and Related Work

Introducing more features to computers or smart devices brings more vulnerabilities given the amount of code the operating systems use. The modern Linux kernel consists of over 26 million LoCs, while Android OS consists of 12-15 million. Naturally, it is not easy to analyze such large bases. Vulnerabilities are discovered at a rate faster than what security updates can keep up with. Furthermore, the rapid introduction of new features introduces more vulnerabilities causing more risks in such massive systems. Therefore, mobile systems utilize another security measure for sensitive operations, namely Trusted Execution Environment (TEE). Android uses ARM's TrustZone while iOS uses Apple's Enclave. For Android devices, the secure OS is varied by vendor and many versions exist including Google's Trusty TEE [15] and the Open-source Portable TEE (OP-TEE) [15] [78], [79].

As discussed in Chapter 2, the TEEs offer better security countermeasures that can eliminate risks instead of mitigating them in many scenarios. TEEs are the industry standard for processing fingerprint authentication as the sensor is hardwired to the secure world which eliminates the risks of a compromised Rich Environment Execution (REE) leaking the users' data. Other applications include cryptography, wallets, payments…etc. These approaches divide the execution environments in the mobile device into REE and TEE. The REE is massive when counting the code base, while the TEE is reserved for sensitive operations and has a small Trusted Computing Base (TCB). It is critical that any TEE service design be generic and does not include any application-specific code which allows it to be utilized by multiple developers without having to request special additions by the vendors.

### 6.2.1 QR Payment Methods

QR payment presentation comes in two types, merchant-presented and buyer-presented QR payments. We thoroughly addressed merchant-presented QR payments in Chapters 3 and 4 and offered data integrity assurances with the designs. Figure 6.1 shows buyer-presented QR payments type where the buyer presents their credit card information and the merchant scans that information and sends it over the credit card network for verification and approval for the transaction to complete.

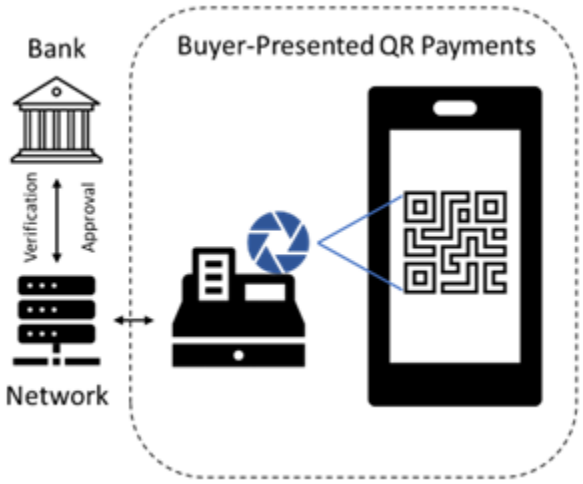



Figure 6.1 Buyer-presented QR payments overview.

Both approaches have their own sets of security risks. If the user's mobile device is compromised, the information they extract from the merchant's QR code can be altered in several places leading to misdirection of the payment and money theft. In [7], this issue was addressed by offering a new data path through the TrustZone that protects the integrity of the processed data with no emphasis on secrecy given the merchant's QR code is not a secret. Furthermore, the decoding process of the scanned QR was addressed in Chapter 4, where a split design was utilized to minimize the code added to the TrustZone. On the other hand, buyer-presented QR codes also have their own risks especially with compromised merchants, where attackers can steal the credit card information from the merchant after the payment is conducted.

Furthermore, the mobile device cannot be trusted to keep the users' credit card information if it is compromised. TruZ-Droid and TruZ-UI addressed the display and presentation portion of this problem by storing the information of the credit card in the TEE, and using secure display mode to present it to the merchant [2], [31]. While useful, it does not address the issues with the merchants' databases if they are compromised, which can still leak the users' data. We addressed protection of the data in Chapter 5, where virtual credit cards were utilized in an offline setting to ensure the real credit card information does not reside in any merchant databases. The delivery of the virtual credit card information was not addressed by any of the aforementioned works.

### 6.2.2 Description of Quick Response (QR) Codes [80] [23] [24]

A QR code is a two-dimensional barcode represented in a bit (or module) matrix with several levels of error correction that allow for versatile information sharing between different endpoints. There are several versions of QR codes ranging from version 1 to 40 with the main differences being in the amount of data each version can carry. There are several other differences in the structure of the modules and the matrix as well [23], [24].

Furthermore, QR codes can easily be customized to include logos in the center of the matrix. The applied error-correction mode will change the amount of data carried for the specific version of the QR code used. More recovery from errors means less data to be carried. There are several factors involved in selection of the version and the error-correction mode which we discuss in sections 6.3 and 6.4. Figure 6.2 shows the structure of QR codes with consideration for various versions and their differences. The mandatory blank space and the finder pattern are what we use to identify that there is a QR code in the image. The alignment pattern allows decoders to adjust the image in cases of skewing in the symbol which allows for conversion into virtual grid representing all the modules and allowing for proper reading of the information contained within.

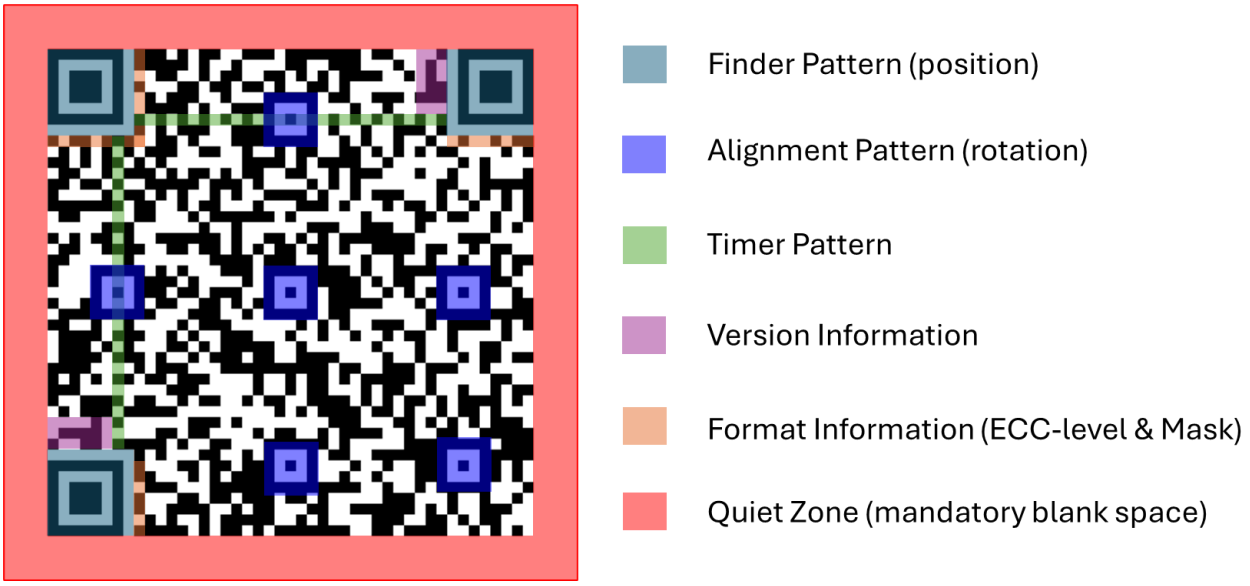



Figure 6.2 QR code structure

The timing pattern is used to determine the central coordinate of which cell in the QR code. The format data (a.k.a. format pattern) contains information about the error correction mode and data mask pattern information. The version data is stored above the bottom-left finder pattern cell and contains information about the version used (if above version 7). Finally, the data and error correction modules represent the rest of the QR code.

### 6.2.3 Frameworks Related Work

Since we are building our framework, we do not expect other designs to offer or relate to the same norms and particulates we address. Two references with some common relations will be discussed. The rest of the review will revisit the three papers produced in this work, namely Chapters 3-5 in order to utilize their outcomes as part of the framework building.

**A. Others Related Work**

**Xianyi Zheng, et. al. [84]** propose a mobile payment framework TrustPAY on TrustZone security enhanced platform, which can ensure payment transactions security and realize privacy friendly payment. They have implemented a prototype system on a simulation environment by using ARM FastModel and Open Virtualization software stack for ARM TrustZone and

presented a development board by using ARM CoreTile Express A9x4. They claim their experiment evaluation and security analysis showed that the scheme can effectively meet the security requirements of a practical mobile-payment with acceptable performance. Furthermore, they claim their system is flexible to support secure applications requiring privacy protection.

They address the security challenges when users disclose sensitive data and privacy information over REE systems and networks. Thus, compromised mobile REE is assumed. Their implementation uses a large portion of the TCB compared to our tendency to minimize the trusted code.

Even though they address privacy they do not address peer-to-peer communication or suggest building SoC devices that can handle full privacy. They do not use rendering UI as strongly established by TruZ-Droid. In addition, the framework does not change the methodology of payment, but utilizes the existing system without protection on the fundamental level like removing service providers or assuming merchant data bases issues or limitations.

In some sense the TrustZone service is mainly to protect the user privacy from the open environment to REE. Our framework operates on deeper security levels and provides wider protection as we shall see in the rest of the chapter.

**Martin Pirker, Daniel Slamanig [85]** claim to demonstrate how privacy friendly payment can be realized using current payment mechanisms in combination with TrustZone enhancements. They discuss the public transport ticket domain as an example. Then they propose a platform framework that can be used for arbitrary applications requiring a privacy preserving online remote prepaid payment system suitable for various payments. They assume the operation through service providers. They suggest this can open the door for issues regarding user's privacy since they disclose sensitive information to the service providers. Their proposal is to

protect users' privacy from public domain breaches. The issue with their model is they still rely on the existing payment system and assume a trust of the service providers. They only add privacy to the user through a system that is not protected from the REE. The TrustZone is just to keep privacy, but there is no protection of privacy from service providers. They do not discuss implementation, and the proposed work is mainly theoretical. In some respect utilizing the TrustZone does not really provide added security, except for storage of sensitive information.

**B. This Thesis Related Components**

In the following we summarize the outcomes of Chapters 3, 4, and 5 and the associated articles to focus on the framework design and utilization.

**Chapter 3 [7]** extended TrustZone functionality to offer robust security measures for specific I/O peripherals, namely, camera and location, to any application on demand. The work mainly ensures integrity of data retrieved by the peripherals. Figure 6.3 shows the invocation mechanism offered in by this design.

Applications that can utilize this functionality include merchant-presented QR payment systems, location attestation for payments and other applications. The work is designed to offer seamless integration for application developers, and transparency to end users. We demonstrated functionality on custom and modified existing applications. The added overhead is within expected margins. The work provides a feasible design for industrial implementations, where the vendors' installed services do not need coordination with potential application developers, and that offers flexibility for both vendors and developers.

A key aspect to this design is that the data secrecy is not of concern given the REE will have access to the peripherals before and after the TEE is done retrieving the data from them. This is by design given the REE needs to have access to peripherals for normal operations, e.g.,

capturing photos or sharing location. The design ensures the integrity of the data is attested by the TEE for security critical operations where the data from the peripheral must not be altered by the REE in any shape or form.

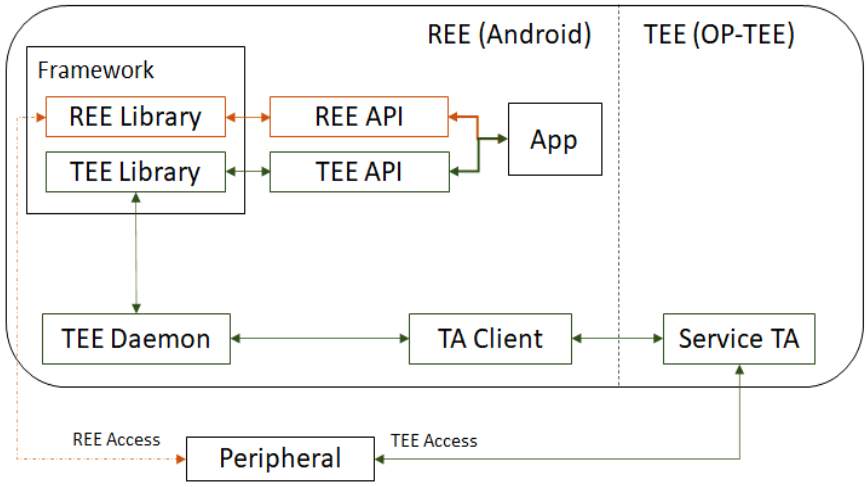



Figure 6.3 Invocation of secure peripheral services between the REE and TEE [7]

**Chapter 4 [8]** designed a new hybrid method by splitting the QR decoder between the normal and trusted worlds. The work compared three different methods. 1) full operation in REE, 2) Split-QR decoding with dynamic and static QR, and 3) server decoding under dynamic and static QR payments. The comparison was meant to highlight the different security features and performance of each approach. The five settings showed the feasibility and advantages of using the Split-QR. Advantages compared to server-based decoders include significant performance improvement and increased convenience for developers, while adding manageable code to the TrustZone.

The success of this application encourages the design of a generalized framework to use split operations, where the TrustZone performs the core critical operations, or delegates them to a dedicated server outside the system if the operations are too extensive. In addition, it manages the meta data that runs noncritical operations on REE.

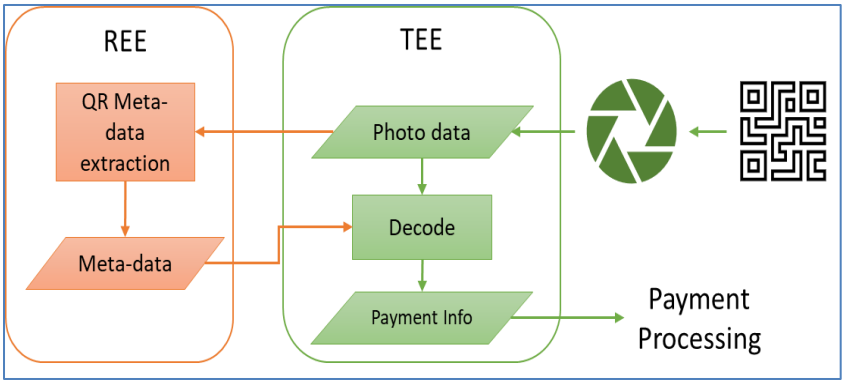



Figure 6.4 Split QR decoder design.

We envision a wider scope of services with large and complex tasks, where the Android system cannot handle them generally. Figure 6.4 shows the design for the split QR decoder where the meta data is extracted by all types of operations can be executed while keeping integrity, and sometimes secrecy too.

Applications can include machine learning classification, biometric data classification, the REE passed onto the TEE uses that information to decode the original picture that was securely captured by the TEE. This relieves a good part of processing from the TEE without introducing risks of the REE manipulating the data given the meta-data is verified with the original photo. After the information is decoded, Split-SSL [2], [7], [31] is used to communicate the data over the network, which ensures integrity across the entire path.

**Chapter 5 [9]** designed a novel virtual credit card generation algorithm, which works offline and under the TrustZone environment. Virtual credit cards can protect users from credit card information theft, which can happen in both physical and digital means. It is common to hear about hacked merchant databases which led to massive leak of credit card information. This risk can be eliminated by the design we set in this work. Combining this design with mobile phones yields high protection for users' data. Meanwhile, it is important that the REE never gets this information since it can be under a remote attacker's control. Hence this is why we designed the

algorithm to work under the TrustZone environment securely. Figure 6.5 shows the design for the offline generation of VCCs in the TEE. We have proven the system's correctness by sending the data to a simulated network to ensure the proper verification of the generated data. In addition, the level of protection can reach near complete security where any fraudulent act can be foiled. Some options can ensure no one can breach the operations beyond intentional bank or user actions in the TrustZone.

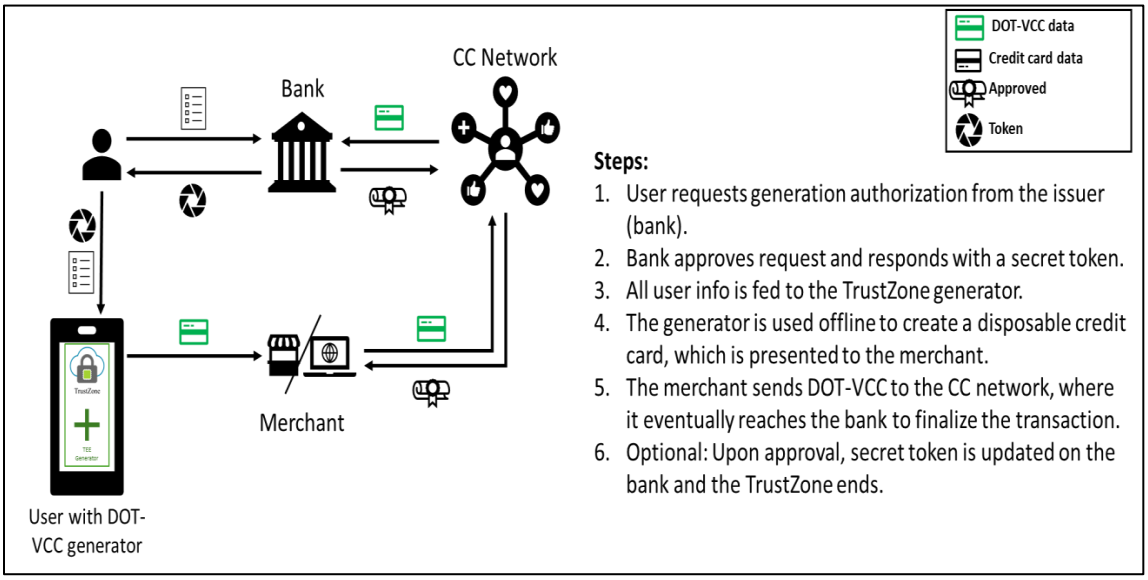



Figure 6.5 Offline virtual credit card generation using TEE.

### 6.2.4 Problem and Objectives

The overall strategy is to provide comprehensive protection with minimal additions to the TEE. We previously addressed three problems; securing IO device, split QR decoding to control data integrity within the TrustZone, and we introduced the concept of offline virtual card generation which eliminates the necessity to share credit card information with merchants and keeps it securely stored in the TEE for generation. Here we extend the problem further to enable the secure transfer of the generated virtual credit card information to merchants, while maintaining the secrecy of the original card information.

### 6.2.5 Threat Model Assumptions

1. Merchant databases are compromised: sellers who scan the user data can be breached and attackers can access users' credit card data from their databases.
2. Android OS on the mobile phone is compromised: an attacker has full root access over the operating system and can perform memory manipulation and management to steal any data that goes in and out of the environment.
3. Service providers are compromised: processing credit cards can be offset to third-party service providers introducing further security risks on users' credit card information.

### 6.2.6 Encoding Functions in The Secure Transfer of Data

Under a compromised OS it is critical not to expose the generated VCC to the REE given a remote attacker can have automated scripts that instantly make use of such data before it is processed by the merchant. Therefore, it is important to directly transfer the data from the TEE to the merchant in a seamless manner that is secure and convenient to the user. Encoding the VCC into a QR code can allow us to display it directly within the TEE by seizing control over the screen buffers and locking the REE out.

During this time, the merchant can scan the data and process it to get the transaction approved. Upon approval, the generated VCC is invalidated and can no longer be used for further transactions. Therefore, any theft of the data that may happen during breaches will not be harmful to the user. Benefits of using an encoder include the following:

1. Immediate presentation of payment information. Encoders are quick and the operation can be done near instantaneously which offers a reliable method of transfer.
2. The encoder uses VCC data, which already completely masks the original credit card data; therefore, it does not expose any important information and acts just as a mediator.

3. The QR code can be treated as an image that is directly displayed using the output buffers in the display driver using the raw image data. This simplifies the process of displaying the data to the merchant.
4. Implementing the encoder within the TEE allows for secure and direct access to the generated VCCs, which allows for more efficient processing that does not require multiple TrustZone SMC expensive calls.
5. An encoder can be designed to be self-contained without reliance on external code libraries to operate given it's primarily tasked with generating a bit matrix, which means the addition of it to the TEE will not introduce risks against the TEE itself.

## 6.3 Solution Design

In this section we discuss two concerns for evaluation, the environment, and the encoder build and communication with TEE and REE. The evaluation covers all components outlined in Table 6.1, and Figure 6.6 show the interactions between the components. Table 6.2 shows the core operation descriptions.

### 6.3.1 Buyer-Presented QR Codes

In buyer presented QR code the seller scans the payment information from the buyer's device. QR codes come in a variety of versions ranging from very simple version 1 to the more data encompassing version 40. The core encoding method remains the same while the details differ [81] In essence, the higher the version the more data can be packed into the QR and the more tolerant to errors it will be. There are, other types of QR codes specialized for applications by developers as seen in WhatsApp [82] for example. Choosing a QR version is application specific, therefore any support for QR encoding or decoding must encompass all versions of QR codes and be accompanied by criteria necessary for selection based on the application. The main

criteria include the size of data to transfer and the number of tolerable errors during the transfer. Increased tolerance to errors means less capacity for data within the specific version. There are four levels of error correction: low (L), medium (M), quartile (Q) and high (H) which allow for 7%, 15%, 25% and 30% of data restoration, respectively.

Using phones to display QR codes is generally desirable given screens have high quality and even cracks and broken displays can be mitigated by the error correction of the QR code itself. Generally speaking, credit card payments do not exceed 1kilobyte of data in total. This makes version 25 a suitable option given its capacity of holding 1269 characters of data leaving enough room even for H-level error correction.

The payment information stored within the TrustZone needs to remain there. The VCC generator allows for a one-way encryption of that data into a one-time use card that automatically deactivates upon usage. That significantly reduces risks carried when buyers present their information to merchants given the VCC data can be leaked after the fact causing no harm to the original data stored in the TrustZone.

The threat model changes when considering an active attack happening on the phone's REE given that such access can be automated to steal the VCC data at the moment of creation and use it before processing by the merchant, which causes a decline for the authentic payment, and more importantly theft of buyer's money. Strictly speaking the REE should never access the VCC information. It is essential to generate it and send in directly from the TrustZone. Two methods of protection against this type of compromise can be considered: 1) elimination of exposure of the VCC data to the REE, and 2) introducing transaction specific info in the presented data.

The first approach entails a full containment of the VCC generation as well as presentation within the TEE. The REE is completely blind after requesting transaction processing from the

TEE as the original data is stored in the TEE, and the generated VCC is created in the TEE, and finally, the QR code is encoded in the TEE and displayed on the phone's screen with secure mode enabled, eliminating the REE's access even to the screen's frame buffer. The TrustZone design allows for acquisition of shared resources and the ability to lock the REE out of using such resources. ARM uses ports that write to the screen buffer and those can be securely mapped into the TEE, then the QR is presented, and the buffers are flushed before relinquishing control over them back to the REE. The second approach for the bank to check further validations of the merchant. However, this can complicate the bank protocols and it is not recommended to follow.

### 6.3.2 Design and Evaluation Plan of the SEEM Framework Integrated System

The design includes integration with the previous components as presented in references [3], [7], [8], [9].

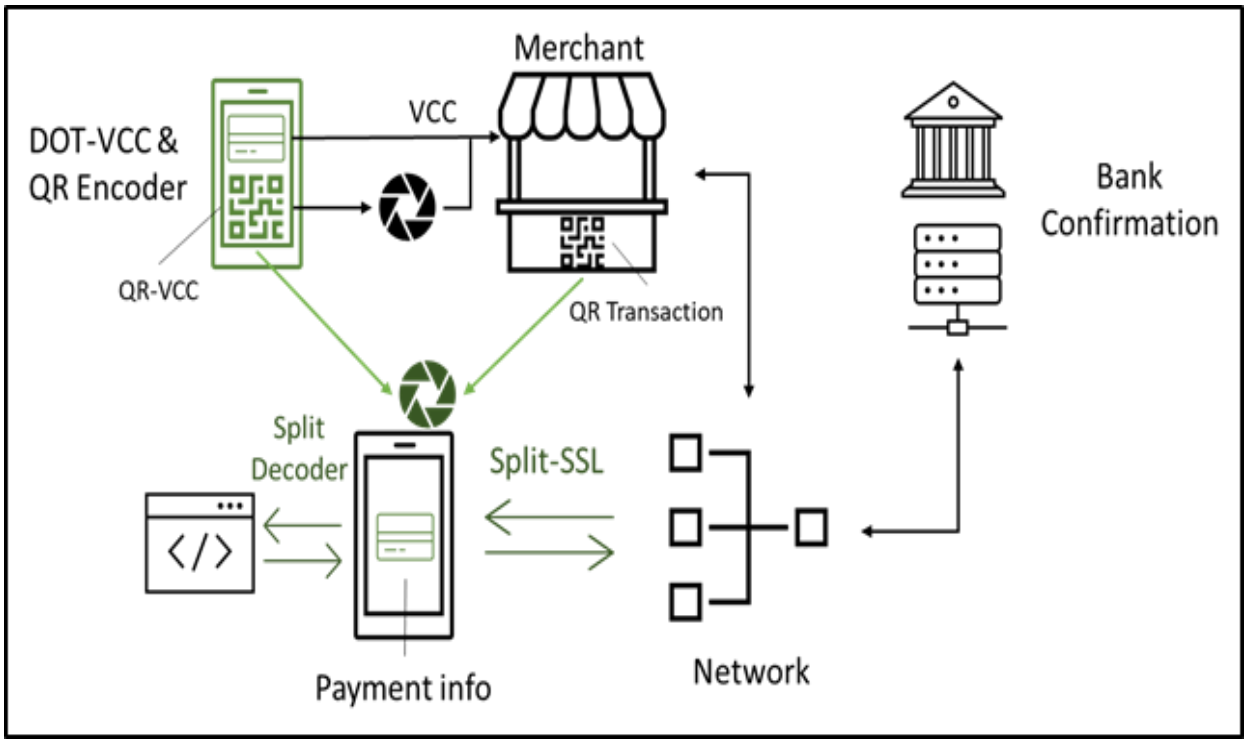



Figure 6.6 SEEM framework integrated system components and their interactions: 1- camera, 2- Split decoder, 3- encoder, 4- DOT VCC

The integrated system is composed of three core transaction analyzers and generators (camera and Split-SSL are treated as one core components. Figure 6.6 shows the integrated system and components with the interaction paths. Table 6.1 shows the functions and their size in the TrustZone. These functions comprise the integrated system proposed. Table 6.2 shows the operations where the first is the sender and the second is the receiver. The table shows the feasibility, type of protection, and relevant players and medium. Evaluation is based on executing the operations.

Table 6.1 Core functions and their LoCs in the TrustZone or TrustProvider. Camera and split are treated as one system.

| Function | Type | TrustZone LOCs |
|---|---|---|
| Camera | receive | 500 |
| Split QR Decoding | receive | 3700 |
| Encoder | send | 950 |
| VCC | send | 300 |

Table 6.2 The core operations where the first is the sender and the second is the receiver.

* 1 = offers integrity assurances only. 2 = offers integrity and secrecy assurances.
** 1 = TrustZone + TrustProvider. 2 = TrustZone + TrustProvider + Bank

| | Sender- Receiver | Protection* | Trust Medium** |
|---|---|---|---|
| 1 | Encoder → Camera | 1 | 1 |
| 2 | Encoder → Camera + Split | 2 | 1 |
| 3 | VCC-Encode → Camera | 2 | 2 |
| 4 | VCC Encode →Split | 2 | 2 |
| 5 | VCC →Bank | 2 | 2 |

## 6.4 Implementation and Evaluation

### 6.4.1 Implementation

A good aspect in an implementation that involves the TrustZone is the size and facing possible vulnerabilities in what should be a secure enclave of data. It is important to minimize the

installed code in such enclaves to ensure easy review and debugging. This includes reduction in the code's size (LoCs added), or eliminating unnecessary dependencies on other external libraries, and minimizing the code's interactions with the REE in order to ease the analysis. We have applied all these aspects in our implementation.

To minimize the amount of code going into the TEE we examined the exact functions and omitted unnecessary code. This elimination was tricky considering that offering more utilities provides better services. often means wider support of functions that are useful. However, it is not the priority in the TEE to have environments, but rather to have only the necessary ones for a specific goal, to secure creation and presentation of payment information to merchants. The necessary functions for encoding the payment information include support for numeric and alphanumeric encoding; binary encoding; and support for segmented information.

The elimination of any unnecessary external dependencies. The OP-TEE defines C99 as the standard C library supported. This means additions to C since they are not supported. We avoided heavy referral to external libraries that would necessitate including them into the TEE, which would grow out of proportion and renders the TrustZone in conflict and introduce many bugs for no critical reasons for our encoder to function. The code was reduced by minimizing the interaction between the Trusted Application (TA) and the TA client on the REE side. The information communicated by the REE include:

1) A reference to the credit card to be used: this reference does not contain information about the credit card itself, it is a public reference that the TEE can expose to the REE while the credit card information remains secure in the TEE. This is similar to how fingerprint authentication works where each fingerprint gets its own reference, and the REE stores the references of accepted fingerprints while the TEE actually verifies the inputted fingerprint matches the securely stored

ones and then returns a reference to the one found. This allows the user of the REE to use a set of credit cards instead of just one.

2) The operation mode: this mode allows the user to select if they wish to utilize a QR transportation method or other methods if necessary. For instance, NFC is another method of transportation desired by different users. The input from the REE is just a flag indicating which mode to be used and is translated by the TEE's API to the proper function.

The REE receives no data from the TEE once the TA call is initiated. The phone switches to the secure mode and the REE is cutoff completely which is necessary to protect the privacy as well as the secrecy of the transaction.

Initially we installed our QR encoder on a normal development machine (x86) to verify the functionality of the encoder and perform the analysis necessary to migrate that implementation for ARM TrustZone. The only standard C dependencies we used were (assert, limits, stdlib, string, stdbool and stdint). All of which are provided by the C99 standard and come as part of the OP-TEE API. The total Lines of Code added to the TEE for the encoder itself were 950 LoCs. We did not implement a new TA for the encoder given that the functionality is integrated with the VCC generator TA and compliments it. The grand total LoCs supporting both the VCC generator and the QR encoder is ~1250LoCs which is reasonably small.

### 6.4.2 Performance of Integrated Services

We have conducted the tests as outlined in Table 6.3. The most detailed and highly needed part is the VCC encoding. In order to assess the performance impact of the system compared to non-TrustZone enabled payment systems, we need to define the time parameters. Table 3 describes and the different time measurements necessary to assess the impact. For certain transactions, the total time needed for generation, sending, and decoding using our system components is given in

Table 6.4. Using this configuration one can envision peer to peer designs where a group of users can use the four functions in their communications. The non-split camera decoding method using nx time to transfer data to external end server for decoding, the transfer time is pretty significant given the data can be quite large. When the QR code is decoded using our system split-decoder design, the transfer to the server only contains small amount of data that does not depend on the size of the captured image. This saves significant amount of time [8]. Table 6.4 shows the detailed functions and performance of the various components.

Table 6.3 Time measurement variables and their descriptions

| Variable | Description | Typical time values (ms) |
|---|---|---|
| C | Time to capture an image from camera | 10-20 |
| E | Encoding time | 50-350 (depending on QR version used) |
| D | Decoding a QR code | 20-250 (depends on image size) |
| T | Split-QR Decoder TrustZone overhead | 15-50 (depends on image size) |
| S | Split-SSL overhead | 350-400 |
| V | VCC generation time | 2-3 |
| VE | VCC generation and encoding = V + E | 50-350 (depends on QR version) |
| O | TrustZone SMC overhead | 200 (depends on QR version) |
| K | TruZ-Droid Keyboard TA overhead | 150-200 |
| U | TruZ-Droid UI overhead | 50-80 |
| Z | External composition time | - |
| x or nx | Network transmission/packet or Communication/set of n packets | Depends on network |
| A | For simple reading: A = C + D + U | 80-350 |

When the split-decoder and the VCC encoder are combined, we can introduce a peer-to-peer system where one device can generate and present a QR code from the TrustZone in VE + O time, while another device uses the split-decoder method to capture an image of the presented QR and process the payment through it for a total time of A + T + x + S.

Note that the VCC-encoder does not rely on any network traffic to generate and present the payment information, which significantly improves both security and performance compared to systems that need external approval for the VCC generation even when the TrustZone SMC overhead is factored into the equation.

Table 6.4 The integrated system core operations, where Dev1 is the sender and Dev2 is the receiver, using TrustZone or standalone TrustProvider for protection.

| Dev1 → Dev2 | Compose | Read | Decoding | Server Comm. | Attestation | Total |
|---|---|---|---|---|---|---|
| Ext/Enc→Cam | Z/E | C | D | nx + S | U | Dev1 = Z/E<br>Dev2 = A + nx + S |
| Ext/Enc→Cam Split | Z/E | C | D + T | x + S | U | Dev1 = Z/E<br>Dev2 = A + T + x + S |
| VCC+Enc → Cam | VE | C | D | nx + S | U | Dev1: VE + O<br>Dev2: A + nx + S |
| VCC+Enc → Cam-Split | VE | C | D + T | x + S | U | Dev1: VE + O<br>Dev2: A + T + x + S |
| VCC → Bank | V | C | D2 | - | U | Dev1: V + O<br>Dev2: C + D2 + U |

## 6.5 Conclusions and Ongoing Work

This work concludes the cycle of development to build an integrated system for transactions of various types with high levels of integrity and security. The cycle covers testing of feasibility and performance. One can demonstrate the overall functionalities of the components: the new communication lines for peripherals (camera and location), split QR for data integrity decoding as an alternative to server-based expensive decoding. The VCC plan includes integration with encoding and decoding options that ensure data integrity and security, in addition to extending the applications with minimum android changes and flexible developer strategies. All of the

components are driven by the TEE operating securely on the TrustZone. In this section we present a summary of all developed components as part of the integrated system. Figure 6.6 shows the layout of the modules including the encoder serving other operations. We also discuss the concept of the TrustProvider, a separate design that integrates all of the components in one device that isn't dependent on Android mobile devices and their infrastructure.

### 6.5.1 Conclusions and Generalizations

The encoder fits within a set of components designed for a comprehensive framework of protection for payments using mobile devices. The previously developed components paved the way for the encoder to complete the pieces and introduce new benefits of its own. For example, it offers a reliable, secure, and seamless method of transfer of generated data to merchants that organically goes with users' expectations from such a service. It also allows for **peer-to-peer transactions** between users where one can use the encoder to display their temporary payment information and the other user can scan that using [7] and process the information using [8] to complete the transaction. Table 6.5 outlines our design highlights compared to other frameworks. All is done while secrecy of both users remains protected from their respective REEs, and from each other without using third-party service providers, or middlemen, to process transactions. This end-to-end security can significantly reduce credit card theft and save users money and effort combating theft.

Table 6.5 Design highlights and comparison summary

| Design | Highlights | Comparison to our design |
|---|---|---|
| SEEM-FTF | Servicing a new payment system using the current infrastructure providing data security and integrity and removal of middle man influences including service providers plus peer to peer transactions and dynamic offline VCC including decoding and encoding and peripherals and utilization on TrustZone based devices or on a standalone TrustProvider. | N/A |
| TrustPAY Framework [84] | Addresses vulnerabilities present on the mobile devices and offers protection against attacks that aim to steal data from the phone or redirect financial transactions to malicious destinations. | Does not address compromised merchant/provider databases. Assumes trust in existing infrastructure. Introduces large code into the TCB. Does not offer peer-to-peer transaction security. Does not support a dedicated TrustProvider implementation. |
| Martin Pirker, Daniel Slamanig [85] | Addresses prepaid payment systems where user privacy is compromised due to service providers. Uses public transport ticket domain to reduce amount of data shared with service providers | Does not offer protection against provider-related breaches. It is primarily concerned with privacy, not integrity nor secrecy of sensitive financial information. Does not address compromised REEs and attacks against mobile devices themselves. |

### 6.5.2 System Integration & Technology Innovation

Using the integrated system design, it allows the development of a standalone TrustProvider that offers integrity and security, while being capable of communicating with TrustZone-based solutions. The integration is built on the design of DOT-VCC system [9] The integrated payment system includes QR encoding and decoding as described in Table 6.2 and Figure 6.6. The system does not require providing critical information to go through the merchant database or service

providers which can be compromised. In addition, the design provides utility to secure buyer-presented QR codes for static or dynamic encoding. Table 6.6 shows a description of using the TrustZone through android devices, and direct standalone TrustProvider.

Table 6.6 TrustZone protection through two methods

| Method | Description | Pros and Cons |
|---|---|---|
| TrustZone in mobile devices | System is installed on the mobile TrustZone. Needs installation permission from manufactures. | Needs personal device compatibility and readiness of mobile manufactures to install services on the device. Hard to ensure REE safe environment. |
| Standalone TrustProvider | System is independent and can execute all transactions using the TrustZone TEE. | Secure user transactions with no middleman. No restriction from TrustZone manufacturers. TEE operation ensures security. |

Implementing either mode, Standalone TrustProvider or TrustZone, can offer better protection. We have demonstrated the system's accuracy and security by sending data to a network to ensure proper verification of data generation. In the following we outline the ongoing work to build the integrated system and the innovations involved.

**A. Our Solution: DOT-VCC Generation System vs Tokenization** [69], [70], [71].

Initiation**:** User requests generation authorization from the bank once, for approval with a secret token. All user info is fed to the TrustZone/TrustProvider generator at the user end.

Transactions**:** Generator creates offline VCCs based on secrets, UTC timestamps and other dynamic parameters, and presents it to the merchant, who sends for the bank approval.

**B. World Infrastructure Countermeasures Limitations.**

1. Threat Model: credit card info leaks through attacks against merchants and 3$^{rd}$-party service providers. Vulnerabilities: Merchant databases, compromised mobile devices, and VCC and verification providers.

2. For card and merchant Tokenization: User or merchant initiates the token. Provider translates token-to-from-CC info. Vulnerabilities: Merchant database or tokenization provider security is not assured for online services, in addition to their own compromises. Users' devices are compromised.

3. For VCC w/Tokenization VCC is generated by a provider. Vulnerabilities: Works online only. Users' devices are compromised. Service provider is not eliminated.

**C. Innovations**

1. Offline VCC generation is done without external VCC providers. VCC is validated only for a time window. Works for online and in-person purchases. Middleman or user device compromises are eliminated.

2. Reduces dependence on 3rd-party infrastructure by offering end-to-end one-way encryption between bank and user. The method mitigates risks present in methods like tokenization, regular credit cards, and others.

3. The VCC generation is based on secrets, timestamps, and other dynamic parameters. The method utilizes 1-way encryption to generate data that is invalidated after use which renders online databases breaches near harmless (with the exception being the bank's database).

4. The standalone TrustProvider: realizing the complexity of dependence on manufactures we added a TrustProvider compatible with the TrustZone.

5. Integrated trust service: Combining transaction formats like QR or NFC with the DOT-VCC integrates system uses including peer-to-peer transactions.

**D. TrustProvider-SoC**

1. The SoC would cut attack surfaces on the communication line and cyber security.

2. TrustProvider would be compatible with the TrustZone/TEE.
3. Setting design parameters and building the prototype, The SoC controls functionalities dedicated to secure transactions.
4. Applicability and compatibility. The design should be flexible to improvements.
5. Open source at the end for independent developers to enrich evolution.
6. Software development and architecture need special settings.

**E. Testing and Software Development**

1. Test relevant attack surfaces and foil the threat model assumptions.
2. Computer development: building a system that operates securely with minim size and threat surfaces.
3. Compatibility: make it portable to other systems, and adoption by manufactures
4. Potential interested stockholders can be invited to conduct real-time testing of all possible attacks through selected customers.

**F. Integrated Trust Service**

1. A general framework unifying QR coding-decoding, and NFC transfer line, can enrich capacity to serve other nonfinancial transactions.
2. Peer-to-peer service can secure independent operations with no middleman or third parties. That would encourage developers.
3. Development of the system through peer-to-peer action and open-source publishing.
4. Activities include integrating encoders, decoders, and cameras to enable conducting merchant operation or VCC based transaction encoding.

# Chapter 7
# Conclusions & Future Work

# Chapter 7: Conclusions and Future Work

## 7.1 Conclusions

### 7.1.1 Assured Data Integrity of Camera and Location Devices (Chapter 3).

Utilizing ARM TrustZone and TruZ-Droid allows a platform to secure several I/O operations. Peripherals include camera, location sensor, Wi-Fi, NFC, and others. Operation security falls in two main categories, namely Security and Integrity. Ensuring security automatically includes integrity. However, integrity does not require secrecy. Our problem was mainly focused on integrity of such operations.

This work demonstrates the TrustZone can be extended to offer robust services to third-party applications without the need for developers to coordinate with vendors. We have built a system that ensures integrity of data collected from peripherals, namely, camera and location, and maintained integrity of data all the way to the server. The added overhead is acceptable when compared to normal untrusted operation. The added service does not introduce inconveniences for user interaction given the transparency of flow, or for applications developers by requiring minimum changes to accommodate the provided TEE functionality.

Our work provides means for extensions to other peripherals in similar manners. It is feasible to secure Near-Field-Communication (NFC) since it is similar conceptually but with different implementation details. Industrial implementation is feasible through the design outlined and there is no need for coordination with applications developers.

### 7.1.2 Split-QR Decoding Hybrid Design for ARM TrustZone (Chapter 4).

1. In this part, we introduced a new hybrid approach to solve the QR decoder problem. The design solution makes it much easier for developers to incorporate into existing applications compared to our previous approach when the server decoded images instead. The overall

results show less overhead with increased transparency and seamless integration. Therefore, this new approach is quite advantageous.

2. The simulation environment was quite sufficient to test the new design given it relies on existing independent components produced by TruZ-Droid and our previous work. This new decoder design can be extended beyond QR payments and towards anything that requires trusted QR code analysis where integrity of data is crucial. Security would be much harder to achieve in the given boundaries. The camera is available to REE at all times except when TEE needs it. Therefore, a QR code displayed on another screen will be hard to keep away from a compromised REE's reach. This is why our hybrid approach works well and makes a difference under the assumption that REE is compromised.

3. Extensions to other wider applications is feasible. We discuss these points in the future work.

### 7.1.3 TrustZone DOT-VCC Generation for Financial Transactions (Chapter 5).

In this part, we present a novel design for virtual credit card (VCR) generation algorithm which securely works offline within the TrustZone environment. VCRs can protect users' cards information from theft. Merchant databases are often hacked resulting in massive leak of credit card information. This design can completely eliminate the risks. In addition, we secured buyer-presented QR codes for static or dynamic encoding. This extension is widely used in some parts of the world like China. Implementing this design with mobile phones or other operating systems can provide full protection. It is important that the normal-world operating system never gets this information considering the high probability of compromise by remote attackers. We have proven the system's accuracy by sending the data to a simulated network to ensure the proper verification of the generated data. The protection can reach near complete security where any

fraudulent act can be foiled. Some options can ensure no one can breach the operations beyond intentional bank or user actions in the TrustZone.

The offline nature of the algorithm cuts off many attack surfaces and windows. Furthermore, the algorithm does not impose any architectural changes to the existing credit card network. The only modifications required are at the very endpoints of the network (the cardholder and issuer server) while everything in-between remains the same. This system uniquely combines the benefits of VCCs and user-end tokenization systems and other parameters. It works with in-person transactions (unlike VCCs), does not require changes to any infrastructure (unlike tokenization), and it is compatible with merchant-end tokenization.

The construction method of the credit card number is based on secrets, original credit card data, and UTC time. The method uniquely combines many existing technologies that yield the unique levels of security and services. Strong hash encryption is used and any change in the card information blocks its use in transactions. The system uses UTC as the dynamic component of the VCC variability. This allows transactions within the same time window if assigned for single or multiple transactions.

The generated VCCs are irreversible and hence inaccessible to the Android or other OS and cannot be utilized by database attackers. The issuing bank and the TrustZone are the only parties who can reconstruct the number. The unique offline VCC generator extends their use to in-person transactions unlike traditional online-only role. The dynamic and offline nature allow the generator to work completely independently from online communication to any remote servers.

The strong security does not yield significant overhead nor introduce risks to the TrustZone environment. The algorithm is robust even in the worst-case scenario where both the payment

terminal and the REE (Android) are compromised by attackers. In addition, the algorithm is convenient, straightforward, and fully safe.

The algorithm can be utilized by many banks and card issuers without specific settings for any of them. This allows utilizing the same TEE service without having to contact devices vendors to support their own specific services. The solutions provide near complete protection with seamless activity. The design is flexible for further improvements and applications on other system or devices beyond mobile systems.

### 7.1.4 SEEM Framework with DOT-VCC and other functions (Chapter 6)

**A. DOT-VCC Generation System**

**Initiation**: User requests generation authorization from the bank once, for approval with a secret token. All user information is fed to the TrustZone/TrustProvider generator at the user end.

**Transactions:** Generator creates offline VCCs based on secrets, UTC timestamps and other dynamic parameters, and presents it to the merchant, who sends a request for the bank approval.

**Countermeasures**: Offline generation is done without external VCC providers. VCC is validated only for a time window. Works for online and in-person purchases. Middleman, databases. or user device compromises are eliminated.

**B. World Infrastructure Countermeasures Limitations.**

1. Threat Model: credit card info leaks through attacks against merchants and 3rd-party service providers. Vulnerabilities: merchant databases compromised mobile devices, VCC and verification providers.

2. For card and merchant Tokenization: User or merchant initiates the token. Provider translates token-to-from-CC info. Vulnerabilities: Merchant database or tokenization provider security

is not assured for online services, in addition to their own compromises. Users' devices are compromised.

3. For VCC w/Tokenization VCC is generated by a provider. Vulnerabilities: Works online only. Users' devices are compromised. Service provider is not eliminated.

**C. Technology Innovations**

1. Offline VCC generation is done without external VCC providers. VCC is validated only for a time window. Works for online and in-person purchases. Middleman or user device compromises are eliminated.
2. Reduces dependence on 3rd-party infrastructure by offering end-to-end one-way encryption between bank and user. The method mitigates risks present in methods like tokenization, regular credit cards, and others.
3. The VCC generation is based on secrets, timestamps, and other dynamic parameters. The method utilizes 1-way encryption to generate data that is invalidated after use which renders online databases breaches near harmless (with the exception being the bank's database).
4. The standalone TrustProvider: realizing the complexity of dependence on manufactures we added a TrustProvider compatible with the TrustZone.
5. Integrated trust service: Combining transaction formats like QR/ NFC with the DOT-VCC integrates system uses including peer-to-peer transactions.

**D. Standalone TrustProvider-SoC**

1. The SoC would cut attack surfaces on the communication line and cyber security.
2. TrustProvider would be compatible with the TrustZone/TEE.
3. Setting design parameters and building the prototype, The TrustProvider-SoC controls functionalities dedicated to secure transactions.

4. Applicability and compatibility. The design should be flexible to improvements.
5. Open source at the end for independent developers to enrich evolution.
6. Software development and architecture need special settings.

**E. Testing and Software Development**

1. Test relevant attack surfaces and foil the threat model assumptions.
2. Computer development: building a system that operates securely with minim size and threat surfaces.
3. Compatibility: make it portable to other systems, and adoption by manufactures
4. Banks will be invited to conduct real-time testing of all attacks and supervise. This can be conducted after system validity.

**F. Integrated Trust Service**

1. A general framework unifying QR coding-decoding, and NFC transfer line, can enrich capacity to serve other non-financial transactions.
2. Peer-to-peer service can secure independent operations with no middleman or third parties. That would encourage developers.
3. Development of the system through peer-to-peer action and open-source publishing.
4. Integrating encoders, decoders, and cameras to enable conducting merchant operation or VCC based transaction encoding.

## 7.2 Ongoing and Future Work

Following the conclusions of section 7.1 we present here the potential feasibility and challenges to the future and ongoing works.

### 7.2.1 SEEM Framework with DOT-VCC and other functions.

Our research revolves around providing generic TEE services that can be utilized by all developers without contacting vendors. In previous works, we developed TEE services that can securely retrieve camera and location data from the peripherals. Feasibility and performance cycle for fully integrated services. Table 7.1 shows the completed the extension possible in order to complete a fully integrated system that offers all possible operations for a financial transaction framework. The system integration is feasible and can be implemented with minimum changes to the available infrastructures and can be extended to other operating systems and payments platforms.

Table 7.1 Feasibility for fully extended services as outlined in the thesis.

| Service | Feasibility | Type of Protection | TrustZone LOCs | Trust + Approvals Needed |
|---|---|---|---|---|
| Offline VCC Generation | Yes | Total data security and integrity | 500 | TrustZone + Bank |
| QR Encoding for QR buyer and VCC | Yes | Data security & integrity; QR buyer present, | 1250 | TrustZone + Bank |
| Split-QR Decoding | Yes | Data integrity and improved security | 3700 | TrustZone + Vendor |
| Bank VCC Reconstruction | Yes | Full data security and integrity. | 0 | TrustZone + Bank |
| All services | Yes | All of the above as detailed | 5.4k for 2 I/O dev, VCC, encoder + decoder. | TrustZone + Bank |

### 7.2.2 Extending Split Functions for Trusted Operations (Chapter 4).

The success of Split-SSL and the Split-QR decoding as outlined in this work suggests extending the splitting to other applications and functionalities. Five types of different nature arise.

1. Small size services that fully need security. They cannot split and must go to the TrustZone; Examples: GPS through a sensor. We estimate such services to be around 10% of services.
2. Services that can be split into essential to run on TEE, and not essential that is possible to run on REE, with no harm to the basics of the operation or data integrity. They are the main candidates for generalization. Examples: QR decoding. We estimate such services to be around 60% of possible services.
3. Services that are possible to split, but the part for REE execution can harm the operation or beyond the REE capacity. This case can be due to complexity and a need of help from more powerful services. Examples: complex machine learning classification. We estimate such services to be around 10% of services.
4. Services that cannot be split and require outside dedicated services. Either fully TEE or fully external service. We estimate such services to be around 10% of services.
5. Large services that cannot be split and no external service are available. Only TEE or the REE can do them. Such cases are out of the scope definition assuming the Android is corrupted. Example: internet-assisted location attestation. We estimate such services to be around 10% of services.

Without splitting, all service types have to be done through external servers or others. After classification we can reduce the types that need full outside support to maybe 10%, and the split with external extensive use to 20% and around 70% can utilize local splitting or full TEE execution. The splitting then serves 70% of the total load. The others can still benefit from the TrustZone albeit traditionally. In the upcoming work we will address these types and show the benefits to performance, integrity, and sometimes security.

The main point here is that if we use the TrustProvider with the capacity of dedicated hardware-software functionalities and availability of large space and execution power for long programs, the splitting problem and running large programs becomes part of the core system operations.

### 7.2.3 Building a Standalone TrustProvider (Section 7.1.4D) requires the following:

a) It should provide all TrustZone functionalities plus some certain hardwire protection and insistence on secure communication for a good fraction of the operations.

b) Should handle extensive execution with minimum communication through regular online services. This service demands security beyond existing infrastructure, which can harm it.

c) The standalone TrustProvider is our proposal to build the protected trust that use either fully secure communication or minimum communication with no true threats to the data. The hardware is designed to minimize dependence on other services.

d) If the idea succeeds and becomes commercial some people will break the design and our role is to keep track of such breaks as usual, but this is the way how thing work. The advantage is that the system we propose does not use heavy communication hence the risk will always be smaller. In addition, many commands are hardwired, and the software has a watch dog that will always watch the traffic and search it on for regular communication or use fully secure protocols that are hard to break.

e) Compared to the Regular TrustZone The main criteria are trust through minimum operations and communication, but they accept to deal with an REE that is compromised. That is a bottle neck of all TrustZone research on mobiles. If we can get rid of the dependence on the mobile altogether while investing a smaller amount on having the TrustProvider can make life easier. TrustZone hardware already can provide the TrustProvider settings, but that must

go through a service provider or/and manufacturer. In our proposal we want to develop one that provides the same service without the need for the REE service except minimum communication with no risky information

### 7.2.4 Work Plan and Prototype Implementation

**A. Testing and Prototype Construction**

The project's vision is to prove the claims outlined above and build a prototype that beats the challenges. The proposal includes extending the testing to prevent various attacks envisioned and show that all operations are safe and accurate. Comparisons with other mainstream methods will be made as well. Another objective is to build a prototype that can be implemented and would address the following outcomes:

1. Build a thoroughly tested credit system that can pass security breaches to reach a true conclusion about feasibly. This includes detailed testing between a number of users independently (peer to peer) using their own attacks and recording all instances when it succeeded. Any failure breaks the system's integrity and claims. We will also test tokenization, regular VCCs, and QR transactions, including encoding or decoding, against the same attacks.
2. Apply stress tests and difficult challenges that can break other competing payment methods and show whether the new system succeeds against all of them.
3. Design a system on chip (SoC) TrustServer that can be programmable to deal with the various payment systems considerations. This device will be designed through the project funding, but the manufacturing contract is part of the commercialization plan. The specifications of the circuit will depend on the use cases and scope of applications. We envision that to include virtual generation, standard credit card payment transaction, QR

encoding and decoding transactions, and possibly communication of high values secure communications. The system can be run standalone or with a network.

4. Produce a prototype that has a well-documented record of foiling all breaches that could affect other established methods. The prototype would work offline or communicate with other systems through defined protocols.

### 7.2.5 Challenges

**B1. Making the daunting job of testing all attack surfaces**: connected with mobile usage vs using the TrustProvider. (Sec7.1.4A-1.4E).

**B2. Building the Integrated Software (Sec 7.1.4F).**

That Can Utilize all Functions. Integration requires a good zone that combines the 4 functions. For encoding, it seems normal, but for decoding that opens a range of challenges if the messages are large or complex. See B4.

**B3. Building the Standalone TrustProvider (Sec 7.2.3).**

Feasible solutions are hard to implement due to three factors. In the following we address these challenges and their answers

1. Challenge: Control by big service providers and manufacturer can skew the advantages of having peer to peer contribution to well-designed TrustProvider. Answer: benefits for the expert developers and the public at large outweigh the controlling tendencies.

2. Challenge: It seems contradictory to offer secure services, while allowing the public to participate in building such structure. The catch is making it available for peer to peer, where a select group can decide to do their work on their own. The hardware design should ensure full protection conditions. Answer: If the users do not preserve such conditions that will be their problem. However, a majority can come with solutions that meet the requirements.

3. Challenge: The third reason has to do with the tradition in security setting threat models that are near impossible to avoid except if you do nothing or isolate. Answer: In such cases isolation is the solution, and there is a reason and feasibility to do so. Isolation does not mean cutting all communication, but keeping only what is needed. There is a good space in optimizing isolation. The proposed framework addresses this point seriously.

**B4. Handling Decoding and Large Programs (Section 7.2.2).**

Extending split functions for trusted operations. The success of Split-SSL and the Split-QR decoding suggests extending the splitting to other applications and functionalities. Chapter 4 and section 7.2.2 discussed this problem in detail.

Without splitting, all service types except the first have to be done through external servers or others. After classification we can reduce the types that need full outside support to maybe 10%, and the split with external extensive use to 20% and around 70% can utilize local splitting or full TEE execution. The splitting then serves 70% of the total load. The others can still benefit from the TrustZone albeit traditionally. In the upcoming work we will address these types and show the benefits to performance, integrity, and sometimes security.

The main point here is that if we use the TrustProvider with the capacity of dedicated hardware-software functionalities and availability of large space and execution power for long programs the splitting problem and running large programs becomes part of the core system operations.

# References

**References Chapter 1**


1. NVD: What is the National Vulnerability Database? Lacework. Available online: https://www.lacework.com/cloud-security-fundamentals/nvd-what-is-the-national-vulnerability-database. 2024
2. Kailiang Ying, Amit Ahlawat, Bilal Alsharifi, Yuexin Jiang, Priyank Thavai, and Wenliang Du, "TruZ-Droid: Integrating TrustZone with Mobile Operating System", MobiSys '18, June 10–15, 2018, Munich, Germany. http://www.cis.syr.edu/~wedu/Research/paper/mobisys2018_truzdroid.pdf
3. Holding, A , "ARM Security Technology Building a Secure System using TrustZone® Technology. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC 009492C_trustzone_security_whitepaper.pdf
4. Section 3 ARM TrustZone - OPTEE- GLOBAL PLATFORM. https://optee.readthedocs.io/en/latest/architecture/globalplatform_api.html
5. Yousra Aafer, Xiao Zhang, and Wenliang Du, "Harvesting Inconsistent Security Configurations in Custom Android ROMs via Differential Analysis", 25th USENIX Security Symposium August 10–12, 2016 • Austin, TX https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_aafer.pdf
6. Matthew Lentz, Rijurekha Sen, Peter Druschel, Bobby Bhattacharjee, "SeCloak: ARM TrustZone-based Mobile Peripheral Control", MobiSys '18, June 10–15, 2018, Munich,

Germany

https://www.cs.umd.edu/~mlentz/papers/secloak_mobisys2018.pdf

7. Ammar S. Salman, Wenliang K. Du, "Securing Mobile Systems GPS and Camera Functions Using TrustZone Framework", SAI 2021 Computing Conference, London, United Kingdom (2021). https://link.springer.com/chapter/10.1007/978-3-030-80129-8_58
8. Ammar S. Salman, Wenliang K. Du, "Dynamic Offline TrustZone Virtual Credit Card Generator for Financial Transactions", Future of Information and Communication Conference (FICC), San Francisco, United States (2022). https://link.springer.com/chapter/10.1007/978-3-030-98015-3_65
9. Ammar S. Salman, Wenliang K. Du, "Split-QR Decoder Hybrid Design for ARM TrustZone", Future of Information and Communication Conference (FICC), San Francisco, United States (2022). https://link.springer.com/chapter/10.1007/978-3-030-98015-3_64
10. Ammar S. Salman, Wenliang K. Du. A Framework for TrustZone Encoding/Decoding for QR Buyer-Presented and VCC Offline Generated Transactions, submitted to Future Technologies Conference 2024. (2024)
11. IEEE Security and Privacy W. Ahmed et. al., "Security in Next Generation Mobile Payment Systems: A Comprehensive Survey," in IEEE Access, vol. 9, pp. 115932-115950, 2021, doi: 10.1109/ACCESS.2021.3105450. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9514868
12. Jonathan Reed, Top five cybercrime types, intelligence security, June 5, 2023, https://securityintelligence.com/news/10-billion-in-cyber-crime-

losses-shatters-previous-totals/,
https://www.techtarget.com/whatis/34-Cybersecurity-Statistics-to-Lose-Sleep-Over-in-2020

**References Chapter 2**

13. National Institute of Standards and Technology (NIST). https://www.nist.gov/
14. NIST National Vulnerability Database (NVD). https://nvd.nist.gov/
15. Android Open-Source Project (AOSP), Trusted Execution Environment (Trusty TEE). https://source.android.com/docs/security/features/trusty
16. SANDRO PINTO, Centro Algoritmi, Universidade do Minho, NUNO SANTOS, INESC-ID, **Demystifying Arm TrustZone**: A Comprehensive Survey, Surv. 51, 6, Article 130 (January 2019), 36 pages. https://doi.org/10.1145/3291047 https://www.dpss.inesc-id.pt/~nsantos/papers/pinto_acsur19.pdf
17. David Clark and kc clafy, trust zones A Path to a More Secure Internet Infrastructure, Journal of Information Policy, Volume 11, 2021, DOI: https://doi.org/10.5325/jinfopoli.11.2021.0026; https://www.caida.org/catalog/papers/2021_trust_zones_jip/trust_zones_jip.pdf
18. Kaspersky, About the trusted zone, https://support.kaspersky.com/KESWin/10SP2/en-US/KESWin-10SP2-en-US.pdf
19. Qinyu Zhu, Quan Chen, Yichen Liu, Zahid Akhtar, Kamran Siddique, "Investigating TrustZone: A Comprehensive Analysis", Security and Communication Networks, vol. 2023, Article ID 7369634, 19 pages, 2023. https://doi.org/10.1155/2023/7369634

20. Alberts, J. (2023). The first steps on the zero-trust journey. Retrieved from https://trustedsec.com/blog/the-first-steps-on-the-zero-trust-journey
21. Poonam Dhiman, Neha Saini, Yonis Gulzar, Sherzod Turaev, Amandeep Kaur, Khair Ul Nisa, and Yasir Hamid, A Review and Comparative Analysis of Relevant Approaches of Zero Trust Network Model, 7 2024 Feb; 24(4): 1328. Published online 2024 Feb 19. doi: 10.3390/s24041328 PMCID: PMC10892953, PMID: 38400486 https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10892953/
22. Vince Ricco, Zones of Trust: A New Way of Thinking about IoT Security, 2017 https://www.darkreading.com/iot/zones-of-trust-a-new-way-of-thinking-about-iot-security
23. QR Code Tutorial, © 2021 Thonky.com, https://www.thonky.com/qr-code-tutorial/introduction
24. Matcha Design. (2021). QR code demystified - part 1. https://matchadesign.com/blog/qr-code-demystified-part-1/
25. Matcha Design. (2011). QR code demystified - part 2. https://matchadesign.com/blog/qr-code-demystified-part-2/
26. Matcha Design. (2011). QR code demystified - part 3. https://matchadesign.com/blog/qr-code-demystified-part-3/
27. Matcha Design. (2011). QR code demystified - part 4. https://matchadesign.com/blog/qr-code-demystified-part-4/
28. Matcha Design. (2011). QR code demystified - part 5. https://matchadesign.com/blog/qr-code-demystified-part-5/

29. Matcha Design. (2011). QR code demystified - part 6. https://matchadesign.com/blog/qr-code-demystified-part-6/

30. ISO/IEC 18004:2006 Information technology- Automatic identification and data capture techniques - QR Code 2005 bar code symbology specification [4.0.1] https://www.iso.org/standard/43655.html

31. Kailiang Ying, "Integrating TrustZone Protection with Communication Paths for Mobile Operating System" (2019). Dissertations - ALL. 1018. https://surface.syr.edu/etd/1018

32. Lesniewski-Laas, C., Kaashoek, M.F.: SSL splitting: Securely serving data from untrusted caches. Computer Networks. 48, 763–779 (2005). https://www.usenix.org/conference/12th-usenix-security-symposium/ssl-splitting-securely-serving-data-untrusted-caches

33. Novikov, I. (2024). SSL offloading decoded: What it means for your security health. Retrieved from https://www.wallarm.com/what/what-is-ssl-offloading

**References Chapter 3**

34. Stefan Saroi, Alec Wolman, “Enabling New Mobile Applications with Location Proofs”, Microsoft Research, HotMobile 2009, February 23-24, 2009, Santa Cruz, CA, USA. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/proofs.pdf

35. Mobile Location Confirmation Documentation https://developer.visa.com/capabilities/mlc/docs

36. Sileshi Demesie Yalew, Pedro Mendonc ̧a, Gerald Q. Maguire Jr., Seif Haridi, Miguel Correia, TruApp: A TrustZone-based authenticity detection service for mobile apps”.

October 2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob). http://www.gsd.inesc-id.pt/~mpc/pubs/truapp-final.pdf

37. Sileshi Demesie Bahir Dar Q. Maguire JrSeif Haridi Miguel Correia, "T2Droid: A TrustZone-based Dynamic Analyzer for Android Applications", August 2017 with 86 DOI: 10.1109/Trustcom/BigDataSE/ICESS.2017.243 Computing And Communications (IEEE TrustCom-17) https://www.researchgate.net/publication/318824757_T2Droid_A_TrustZone-Based_Dynamic_Analyser_for_Android_Applications
38. Darius Suciu, Radu Sion, "Droids entry: Efficient Code Integrity and Control Flow Verification on TrustZone Devices". May 2017, 21st International Conference on Control Systems and Computer Science. https://www.computer.org/csdl/proceedings/cscs/2017/1839/00/07968556.pdf
39. Bo Zhao; Yu Xiao; Yuqing Huang; Xiaoyu Cui, "A Private User Data Protection Mechanism in TrustZone Architecture Based on Identity Authentication". Tsinghua Science and Technology (Volume: 22, Issue: 2, April 2017). https://ieeexplore.ieee.org/document/7889643
40. Xianyi Zheng; Lulu Yang; Gang Shi; Dan Meng, "Secure Mobile Payment Employing Trusted Computing on TrustZone Enabled Platforms", August 2016. IEEE Trustcom/BigDataSE/ISPA. https://ieeexplore.ieee.org/document/7847180

41. Hassaan Janjua, Wouter Joosen, Sam Michiels and Danny Hughes, Celestijnenlaan," Trusted Operations on Sensor", Data Sensors 2018, 18, 1364 https://ieeexplore.ieee.org/document/7590042

42. J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, M. B. Srivastava, "participatory sensing", 2006, Center for Embedded Networked Sensing (CENS), University of California, Los Angeles. https://escholarship.org/uc/item/19h777qd

43. Liu, H.; Saroiu, S.; Wolman, A.; Raj, H., "Software abstractions for trusted Sensors", In Proceedings of the ACM 10th International Conference on Mobile Systems, Applications, and Services, Low Wood Bay, UK, 25–29 June 2012; pp. 365–378. https://www.microsoft.com/en-us/research/publication/software-abstractions-for-trusted-sensors/

44. Kamble, P.A. and Neha Patil, SECURE ONLINE PAYMENT WITH ARM TRUSTZONE", International Journal of Development Research Vol. 07, Issue, 07, pp. 13872 - 13875, July, 2017 https://www.journalijdr.com/sites/default/files/issue-pdf/9394.pdf

45. Waqar Anwar, Dale Lindskog, Pavol Zavarsky, Ron Ruhl , "An Alternate Secure Element Access Control for NFC Enabled Android Smartphones", International Journal for Information Security Research (IJISR), Volume 4, Issue 1, March 2014 https://infonomics-society.org/wp-content/uploads/ijisr/volume-4-2014/An-Alternate-Secure-Element-Access-Control-for-NFC-Enabled-Android-Smartphones.pdf

46. Shafiq ur Rehman and Jane Coughlan, "An Efficient Mobile Payment System Based On NFC Technology", International Journal of Computer and Information Engineering Vol: 7,

No: 6, 2013, https://waset.org/publications/7277/an-efficient-mobile-payment-system-based-on-nfc-technology

**References Chapter 4**

47. Chris Lesniewski-Laas and M. Frans Kaashoek, “SSL splitting: securely serving data from untrusted caches”, Proceedings of the 12th USENIX Security Symposium. Washington, D.C., USA. 2003 https://www.usenix.org/legacy/events/sec03/tech/full_papers/lesniewski/lesniewski.pdf

48. Mauro Carvalho Chehab, mchehab, ZBar maintenance and versions, https://github.com/mchehab/zbar

49. Jeff Brown, “ZBar bar code reader”, Copyright 2007-2010 (c) Jeff Brown, last updated July 2015. http://zbar.sourceforge.net/about.html

50. Jeng-An Lin, Chiou-Shann Fuh, "2D Barcode Image Decoding", Mathematical Problems in Engineering, vol. 2013, Article ID 848276, 10 pages, 2013. https://doi.org/10.1155/2013/848276 https://www.hindawi.com/journals/mpe/2013/848276/

51. Nivedan Bhardwaj, Ritesh Kumar, Rupali Verma, Alka Jindal and Amol P. Bhondekar, “Decoding Algorithm for color QR code: A Mobile Scanner Application”, (ICRTIT), 2016 International Conference (IEEE), DOI:10.1109/ICRTIT.2016.7569561 https://www.researchgate.net/publication/309041301_Decoding_algorithm_for_color_QR_code_A_mobile_scanner_application; https://ieeexplore.ieee.org/document/7569561

52. Madeline J Schrier, “Barcode Decoding in a Camera-Based Scanner: Analysis and Algorithm”, PhD Thesis, © Madeline J Schrier 2015 https://conservancy.umn.edu/bitstream/handle/11299/175329/Schrier_umn_0130E_16096.pdf%3Bsequence%3D1
53. Databases collection, “Images of QR Codes: versions 1-4, random four digit numbers Simple QR codes dataset”, https://www.kaggle.com/coledie/qr-codes?select=qr_dataset, “Finder patterns of QR codes - Yolo format, Complex QR codes dataset”, https://www.kaggle.com/samygrisard/finder-patterns-qr-code
54. Android Native Development Kit (NDK). Android Developers. Available Online: https://developer.android.com/ndk
55. Overview of the V4L2 driver framework. Bootlin. Available online: https://elixir.bootlin.com/linux/v4.4/source/Documentation/video4linux/v4l2-framework.txt
56. X Window System open-source project. X.Org Foundation. Available online: https://www.x.org/wiki/
57. ZBar decoder, Wikipedia, https://en.wikipedia.org/wiki/ZBar

**References Chapter 5**

58. Online payment fraud, https://www.ravelin.com/insights/online-payment-fraud, last accessed 2021/08/01.
59. ISO/IEC 7812-1:2017, Identification cards — Identification of issuers — Part 1: Numbering system, https://www.iso.org/obp/ui/#iso:std:iso-iec:7812:-1:ed-5:v1:en, last accessed 2021/08/01.

60. Luhn, H.P.: Self-checking number punch, https://patents.google.com/patent/US2731196A/en, (1956).
61. Calculate CVV/CVC, iCVV, CVV2/CVC2, dCVV for Visa MasterCard, https://neapay.com/online-tools/calculate-cvv-cvc-icvv-cvv2-cvc2-dcvv.html, last accessed 2021/08/01.
62. Resuello, J.: Complete Guide to Virtual Credit Card Numbers, https://www.forbes.com/advisor/credit-cards/virtual-credit-card-numbers-guide/, last accessed 2021/08/01.
63. How Do Virtual Credit Cards Work & How Can You Get One? https://www.creditkarma.com/credit-cards/i/virtual-credit-card, last accessed 2021/08/01.
64. Rubenking, N. J., 5 Things You Should Know About Virtual Credit Cards, https://www.pcmag.com/news/5-things-you-should-know-about-virtual-credit-cards, last accessed 2021/08/01.
65. Rupp, M.: The Challenges and advantages of EMV Tokenization, https://www.cryptomathic.com/news-events/blog/the-challenges-and-advantages-of-emv-tokenization, last accessed 2021/08/01. https://optee.readthedocs.io/en/latest/architecture/globalplatform_api.html, last accessed 2021/08/01.
66. Rubin, A., Wright, R.: Off-line generation of limited-use credit card numbers, https://patents.google.com/patent/US20020073045A1/en, (2002).
67. Molloy, I., Li, J., Li, N.: Dynamic virtual credit card numbers. In: International Conference on Financial Cryptography and Data Security. pp. 208–223. Springer (2007).

https://www.springerprofessional.de/en/dynamic-virtual-credit-card-numbers/2930130

68. Park, C., Park, C.: Public Key based Virtual Credit Card Number Payment System for Efficient Authentication in Card Present Transaction. Journal of the Korea Institute of Information Security & Cryptology. 25, 1175–1186 (2015). https://www.researchgate.net/publication/285647501_Public_Key_based_Virtual_Credit_Card_Number_Payment_System_for_Efficient_Authentication_in_Card_Present_Transaction
69. Yingjiu Li a, Xinwen Zhang b, Securing credit card transactions with one-time payment scheme, Electronic Commerce Research and Applications, Volume 4, Issue 4, Winter 2005, Pages 413-426 https://www.sciencedirect.com/science/article/abs/pii/S1567422305000177
70. Credit card tokenization explained: Everything you need to know – stax. (2024). Retrieved from https://staxpayments.com/blog/credit-card-tokenization-explained/
71. Rourke, T. (2024). The Complete Guide to Payments Tokenization. Retrieved from https://www.aciworldwide.com/a-primer-on-tokens-tokenization-payment-tokens-and-merchant-tokens; https://www.aciworldwide.com/
72. Choh, C. (2023). VCC fuels demand for Private Credit & Fund Tokenization. Retrieved from https://www.ssctech.com/blog/vcc-fuels-demand-for-private-credit-fund-tokenization

73. Merchant Tokenization Merchant tokenization. Retrieved from https://www.payair.com/merchant-tokenization/

74. B2B/B2C eCommerce Forecast, International Trade Administration. Available online: https://www.trade.gov/ecommerce-sales-size-forecast

75. Payment Card Fraud Losses Reach $32.34 Billion. The Nelson Report. Available Online: https://www.globenewswire.com/news-release/2022/12/22/2578877/0/en/Payment-Card-Fraud-Losses-Reach-32-34-Billion.html (2022)

76. Global Ecommerce Sales Growth Report. Shopify. Available online: https://www.shopify.com/blog/global-ecommerce-sales (2024)

77. Top Cybersecurity Statistics for 2024. Cobalt. Available online: https://www.cobalt.io/blog/cybersecurity-statistics-2024 (2024)

**References chapter 6.**

78. ARM Security Technology: Building a Secure System using TrustZone Technology, (2009). Available online: https://documentation-service.arm.com/static/5f212796500e883ab8e74531

79. Trusted Framework Open-source Portable Trusted Execution Environment (OP-TEE). Available online: https://optee.readthedocs.io/ Accessed 2024/02/10.

80. Barcode Technologies, Morovia, https://www.morovia.com/manuals/barcode-dll/shared.bartech.php; QR code payments: How businesses can generate and use QR codes, stripe, 10-2023 https://stripe.com/resources/more/qr-code-payments

81. Nayuki, QR Code generator library, GitHub. Available online: https://github.com/nayuki/QR-Code-generator. Accessed: 2023/12/15.

82. About WhatsApp QR codes, Help Center, WhatsApp. Available online: https://faq.whatsapp.com/2416198805185327. Accessed: 2024/02/20

83. MST Payments: What Merchants Need to Know About Magnetic Secure Transmission. Credit Card Processing, Payment Technology and Hardware. Payment Depot. Available online: https://paymentdepot.com/blog/mst-payments/

84. Xianyi Zheng, Lulu Yang, Jiangang Ma, Gang Shi, Dan Meng TrustPAY: Trusted Mobile Payment on Security Enhanced ARM TrustZone Platforms, 2016 IEEE Symposium on Computers and Communication (ISCC). https://ieeexplore.ieee.org/abstract/document/6296107

85. Martin Pirker, Daniel Slamanig, A Framework for Privacy-Preserving Mobile Payment on Security Enhanced ARM TrustZone Platforms, 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications. https://ieeexplore.ieee.org/abstract/document/7543781.

## Vita

Ammar Salman was born in Ithaca NY, USA. Received a Bachelor of Computer Engineering degree from Alquds University, Jerusalem, Palestine in July 2016. He received a Master of Engineering degree in Computer Engineering from Syracuse University, (Syracuse, New York, USA) in 2021. This dissertation was defended in June 2024 at Syracuse University. And the degree earned in June 2024.