Towards a General Purpose Trusted Computing Platform for All Vendors and Applications

Xiaokang Wang, B.S.

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Intelligent Systems)

Supervisor: Donal O'Mahony

August 2021

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Xiaokang Wang

August 31, 2021

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Xiaokang Wang

August 31, 2021

Towards a General Purpose Trusted Computing Platform for All Vendors and Applications

Xiaokang Wang, Master of Science in Computer Science

University of Dublin, Trinity College, 2021

Supervisor: Donal O'Mahony

The objective of this research is to explore the possibility to create a trusted computing platform that executes untampered programmable logic from multiple uncoordinated software vendors at a reasonable speed while protecting the data protection rights of the users with data usage and transfer transparency.

A specification was created to support all desired characteristics. It defines a manifest that defines the interface of interaction between the trusted computing platform and a programmable executable binary. The trusted computing platform will execute the executable binary data defined in the manifest in the form of Web Assembly in an isolated environment if the invocation request from the trusted computing client satisfies the requirements defined in the manifest. The input and output data is managed by the trusted computing platform, with checking performed on the input and output data based on the invocation request, manifest, and the input themselves. Different type of input and outputs allows the programmable logic to communicate with the trusted computing client using plain text, with other trusted computing platform using session, with users using protected input and output, with tamper-resistant storage with non-violate storage, and receive high-quality random numbers using random.

A proof of concept implementation of this specification is created to evaluate the functionality of this specification. The specification shows it is possible to implement this specification in software with the technologies currently available. A demonstration application is created to showcase the ability of the trusted computing platform to protect the data protection right of the users.

The conclusion of this paper shows that it is possible to create a trusted computing platform that satisfy all the requirements mentioned according to the specification defined with existing technologies.

Acknowledgments

I would like to thank my supervisor Dr. Donal O'Mahony for providing crucial guidance and support for the dissertation. This dissertation wouldn't be possible without swift, patient, and positive feedback at every stage. Thank you very much!

I would also like to thank my parents for their unwavering support and understanding of me. I would like to thanks my friends and classmates, that shares their journey with me.

Finally, I would like to thank those who have lighten my path but I cannot have their name shown with a special hash string that includes my words and thanks towards them:

`|p7nH]rE,|^IH=VVnh&TDosi+?8./u%hH/KsoXI8`

`DfNk:e#O8ga{HXQ@0qRrDtF}U9pM,^W*y'yT+*J`

Xiaokang Wang

University of Dublin, Trinity College
August 2021

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The need for trusted computing comes from the need to protect the information in not only the transfer channel but also on the endpoint where the information is collected and processed. The need to protect data on the endpoint is because misbehaving software and users can access and modify data beyond the control of the software developers. This leads to many software including anti-cheat software tries to prevent users from modifying the game program or accessing game process with invasive and intrusive means including kernel-mode driver and full device check, and often failed to achieve its intended goals while creating significant hindrance to end-users(Lehtonen et al. (2020)) . Currently trusted computing platforms only provide general purpose trust computing functionality to device manufacturers, not individual developers, and cannot protect the interest of the users. There is a need for a new approach that enables any developer to utilize trust computing platforms that protects the interests of users. This paper indented makes it possible to create a device with an improved version of a trusted computing facility.

1.2 Research Question

The question that will be discussed is: Is it possible to create a trusted computing facility that is multi-tenant, general purpose, and promotes user's data protection rights? In this context, multi-tenant means multiple uncoordinated firmware vendors can share a single physical trusted computing device without the need for approval from the original device manufacturer; general purpose meaning the trusted computing device supports all operations and logic in a general way by proving a Turing complete programmable logic; promote user's data protections right means limiting the unwanted transfer and sharing

of user data with system enforced limit on the usage and transfer of sensitive user input data or device sensor data while making it possible to retain the full functionality of the application.

1.3 Dissertation Layout

This dissertation will begin with the introduction of the concept of trusted computing. This includes what is trusted computing? What can it do? And why it is necessary to solve some specific problems in computer engineering? Then, there will be an introduction to the state of the art for currently available trusted computing solutions. This includes the form of these trusted computing solutions, how are they used, and their limitations. This will be followed by an analysis of the cause of the limitations.

After the introduction of previous works, a new specification that attempts to address the question will be described. This will begin with an overall description of the design, followed by a detailed description of important components.

Once the design of the specification is conveyed, the proof of concept implementation of this specification will be described. This includes the method of implementation of the design and demo application that showcases the specification.

The evaluation of the specification will follow the implementation chapter. The evaluation includes the use cases, the possible attack and respective countermeasures, and the limitations of the current specification.

A discussion will follow the evaluation chapter to give an analysis of the possible paths for the adoption of this specification. Possible social impacts associated with the adoption of a trusted computing platform based on this specification are also discussed.

Finally, a conclusion chapter will discuss the finding of this paper. The future works that could assist the further development and adoption of this specification.

Chapter 2

State of the Art

Trusted Computing is a concept popularized by and promoted by the Trusted Computing Group. The Trusted Computing Group is an organization formed to develop and promote open, vendor-neutral standards for hardware-assisted security solutions. It has standardized Trusted Platform Module version 1.2 and 2.0 found within a personal computer, and promoted their adoption. With trusted computing, a computing device will behave in a consistent way that is enforced by both software and hardware(Wang et al. (2013)) .

Trusted Computing devices can store and manage sensitive data safely, and process them according to predefined rules. This allows operations involving sensitive data to be completed on devices possessed by an untrusted party.

2.1 Attack Model

2.1.1 Model

The trusted computing design assumes an atypical attack model(Smith (2013)) (Zhang et al. (2007)) (Bhargavan et al. (2017)) . Usually, only the communication channel is considered to be controlled by an adversary. In the design of the trusted computing platform, both the communication channel and the endpoint device are considered potentially compromised. In addition to injection and malicious modification of data transferred through communication channels, the endpoint device can maliciously read and modify the code and data used by the application in an ordinary execution environment.

2.1.2 Usage

This design considers both the communication channel and users(and/or attackers with physical access to the device) to be adversaries. This allows the trusted computing hard-

ware to act as a representative of a remote party's interest resembling an embassy in the real world.

The trusted computing platforms can be used to facilitate crypto operations with managed keys, shielded key storage, offline electronic currency wallets, and anti-cheating engines (Lehtonen et al. (2020)) (Park et al. (2020)) . These usages all derive their function from the promise that the trusted computing platform will still operate as intended even if the attacker has physical control of both the host and the trusted computing device.

2.2 Capabilities

Trusted Computing devices can be used to complete operations as intended and create proof to remote parties that certain procedures have been completed. For any untrusted ordinary execution environments, it can only prove the knowledge it has, not procedures completed.

Additionally, Trusted Computing makes it significantly more difficult to tamper with the procedure used to process user data even if the software running in the ordinary execution environments is compromised, or if the device itself is physically controlled by an adversary.

2.2.1 Operations yet to be Represented in Cryptography Functions

Proofs in trusted computing platforms are generated after a procedure has been completed inside the trusted computing environment, and can create proof for any procedure. This proof will certify the result of a certain operation like the device was booted into a known status. This is different from ordinary execution environments where proofs are generated with cryptography functions. There are only a limited set of cryptography constructions to represent a finite amount of relationships.

2.2.2 Operations Involve Persistent Storage

Trusted computing platforms have exclusive access to non-replayable storage(like shielded internal non-volatile storage or Replay Protected Memory Block (Reddy et al. (2015))) that can be used to create non-replayable interaction with the trusted computing platform. In contrast, ordinary execution environments have states that can be checkpointed and restored, which means the attacker can replay any interaction indefinitely.

Creating a non-replayable interaction is required to facilitate financial transactions, enforce retry limits for password verifications, and irreversible deletion of sensitive information.

2.2.3 Input and Output Protection

Trusted computing platforms have access to input and output devices. This makes it possible to ensure the data received can be directly delivered to the end-user without interference from software running on the system.

With an ordinary execution environments based information protection system, the information is encrypted when transferred over the network or when stored in a hard drive. The information will remain unencrypted and accessible to privileged software running in the same system. This means if the software on a local device is compromised or has backdoors, it could allow an attacker to gain access to information available to the user without being discovered.

Creating true end to end protection requires the ability to control the input and output devices to prevent compromised software from gaining access to sensitive information during the input collection and output displaying process, as the information will not be passed through a potentially compromised ordinary execution environment.

2.3 Implementations

2.3.1 Smart Card

A Smart Card, and its most dominant subtype the Java Card, is a kind of trusted computing implementation typically used by banks as an authentication device. Initially, the Java Card was designed by Sun Microsystems, which is now a subsidiary of Oracle Corporation.

A Smart Card provides a trusted computing facility to its issuer by giving them exclusive permission to load applications on the card. This card can then interact with the card reader based on the procedures and data stored on the card.

Without the keys available only to the original manufacturers of the cards, applications cannot be loaded onto the card. This prevents an attacker from accessing resources stored on the card, but also stops third party developers from making use of the hardware without approval from the original device manufacturer. It has been effective in creating a trusted computing environment for a single purpose, however, due to its limited resources and the requirement to load programs by the original vendor in advance, it cannot be used as

a general-purpose trusted computing device.

Overall Design

Smart Cards are usually physical devices in the form of a physical card. It will be connected to a host device with an adapter or reader through either 6-pin physical contact (International Organization for Standardization (2011)) (International Organization for Standardization (2007)) or a wireless NFC(International Organization for Standardization (2018)) interface implemented on the Smart Cards.

A physical smart card typically has a set of software components running on it. This includes the Card OS, Native Services, Java Virtual Machine, Framework, API, JavaCard Runtime, and Applets. The Applets are user-programmable components in the Java Card. Other components are used to support the applets in various ways. (Sun Microsystems (1998))

Each card has a set of predefined keys that allow the original manufacturer to load applets into the card. Only applets on the card can access the resources on the card. Since a smart card is a physical device, it has its own separate memory and execution environment that cannot be easily manipulated.

Programming Interface

The programming interface for defining functions of smart cards is usually Java Card API. The Java Card API allows the Applets to be developed. A program running on the Java Card will need to be compiled and converted into a Java Card specific applet format before being loaded onto the Java Card. The loading process requires a special utility to interact with the management applet on the card to load and install the applet bytecode.

The smart card typically has a very limited amount of resources. Most of them have less than 2048 bytes of volatile storage and less than 128 KByte of non-volatile storage. Since the Java Card API is different from the standard Java environment. Most libraries in Java will not work in the Java Card environments, Java card-specific libraries are required for developing on Java Card. A Java Card specific API provides cryptology and management API for Java Card applets. However, the availability of specific cryptology primitives will vary depending on the card model.

The API provided by the Java Card environment includes memory management, cryptology operation, device peripherals, transaction, structure processings APIs. However, some of these APIs are optional. This limits the portability of the code and makes it harder to reuse existing code bases.

The installation of applets is usually divided into 4 steps: Compilation, Upload and Install to Java Card, and finally Personalisation.

Compilation of Java Card applets uses the same compiler as Java code running in a standard Java environment to generate class files. Then, a converter will also be used to complete the translation from a standard Java class file to a processed applet CAP file. During this processing, some of the processing will be completed on the converter to reduce the complexity of code used for interpreting these contents on the card. The processing conducted by converter includes checking if only supported Java language features are used, initializing static variables, resolving symbol names, and creating virtual machine data structures. All these steps make it easier to execute byte codes on the card by offloading some of the initiation tasks to the converter (Chen (2000)) .

After the compilation step, the applet's CAP file will be uploaded to the card. This step is typically accomplished with Global Platform Card Manager. Global Platform Card Manager is an application present on the card that handles the management of applets on the card. To upload an applet to the card, the host needs to be authenticated with the Global Platform Card Manager with keys defined during the initialization of the card. The off card applets CAP file installer will convert the applets CAP file to a set of APDU commands in order to write the applets to the persistent memory of the Java Card. After the data has been copied, the off card installer will invoke the installation procedure to register the applet with the Card OS.

Finally, in the Personalisation step, the applet will be executed and custom commands will be used to upload the necessary information for subsequent usages.

In this procedure, the Java Card prevents an attacker from gaining information on the Java Card by restricting the access of Card Manager to entities holding keys defined during the initialization of the card. This restricts the ability of the device to be useful for third-party developers.

The cooperative installation procedure with converter and Global Platform Card Manager client makes it possible to install applications while keeping logic on the card to the minimum to reduce complexity and cost.

Interaction Interface

The programming interface for interacting with smart cards is usually Personal Computer/Smart Card API. The application in the external environment interacts with the smart card by sending binary commands to it and parses the response from the smart card. Since there is no predefined format for every command. The library usually just needs to support a very limited set of operations. This means the user of the library can

expect the library to support all the commands necessary to interact with the smart card. However, it is the application's responsibility to construct the command and interpret the result. Each command will be formed in the format of Application Protocol Data Unit regardless if the card is connected with a wired or wireless interface. Each Application Protocol Data Unit command has a predefined header structure that will be understood by the interface. However, the exact format of the payload will be defined by the applets on the card.

Smart Cards interact with the host device in a command-response model. The host device sends commands in smart card Application Protocol Data Unit format to applets on the smart card. The applets idle, until receiving a command from the connected host. The applet processes the command and sends back a response. (Sun Microsystems (1998))

Both the request and response format allow the driver and the Card OS to interpret the command to complete their function without limiting the customization potential of applets. Apart from predefined instructions like selecting applets by their applet identifier, all instructions can be defined by the applet itself. The applets can define the instructions and the format of input and output data based on its requirements. This has allowed the developers to create applications on the card based on their use case by making the inputs and outputs generic. The same format of commands is used for both contacted interface and wireless NFC interface.

Typical Application Design

A typical smart card application will be designed as a card applet installed in a smart card issued by the vendor organization. This smart card will interact with applications created by the same organization or through a standardized protocol. Each vendor organization will need to have a separate physical card. The applet on the smart card will accept commands from the reader, checking it against constraints. If the constraints are met, the card will complete the action with cryptology keys stored on the card's memory and return the result to the reader. Typically, the card is usually used to store a small amount of data and complete operations. Since its resource is extremely limited, the amount of logic enforced by the applets on the card is typically minimal.

Bank Cards, GPG Cards are typical Smart Cards. They have a similar function: hold sensitive information and run operations on it without revealing sensitive data itself. Bank Cards are authentication devices that are designed to be unclonable. This is accomplished by embedding a signing key into the card and allows the host device to request a signature when conditions are met. This has made it possible for the user to interact their bank

card with a terminal without the fear of card cloning, which occurs frequently as magnetic strip cards can be copied. Additionally, Bank Cards enforce PIN input attempts limits so that stolen cards are less useful to attackers.

2.3.2 TPM

Trusted Platform Module 2.0 (Mitchell (2005)) (Arthur et al. (2015)) is a kind of trusted computing implementation typically used by desktop computer's operating systems. It is included in most desktop computers and laptops.

Trusted Platform Module 2.0 specification is designed by Trusted Computing Group, and implemented by various security device manufacturers. The 2.0 version of the Trusted Platform Module specification superseded the previous 1.2 version with its significant efficiency and flexibility improvement.

TPM accepts commands from the host device and processes these commands based on its specifications and resources within. Unlike Smart Cards, TPM is not programmable. Different programs can share the TPM since it allows keys to be swapped to external storage. Some advanced yet limited settings are supported to customize the usage condition for TPM managed resources.

TPM has achieved the goal of sharing the same trusted computing device among different vendors, however, it is a fixed-function device with limited functionality that is not generally useful for all applications.

TPM can be used to store unextractable secret keys like smart cards, however, it can handle significantly more keys with swapping mechanics. It can also protect passwords from dictionary attacks with a global lockout rate limiter. Additionally, with usage conditions, it can be used to create full disk encryption that unlocks an untampered system automatically.

Overall Design

TPM can be in the form of a physical chip or an isolated region within the CPU when assisted by a supported motherboard. If it is in a physical form, then it will connect to the motherboard of the desktop computer with a TPM socket.

TPM is not user-programmable, however, it provides a set of APIs to utilize the facilities within it. The API is designed in a way that allows engineers to create trusted computing assisted applications with the facility provided within the TPM. These facilities include hardware managed key storage to create and use an unextractable key to prevent duplication, non-volatile data storage to create non-reversible interaction, a platform configuration register that verifies the integrity of the connected system, and a

policy system that allows engineers to define the specific usages constraints for managed resources.

Each TPM has a limited amount of storage that cannot fit all the user's keys. However, the TPM solves this issue by wrapping the private key with an unextractable key inside the TPM. This makes it possible to make theoretically infinite amounts of keys accessible only to TPM while only storing a limited amount of data within the TPM. This ensures different applications can use the same limited hardware with minimal interference.

In order to make it possible to invalidate the keys managed by TPM, 3 user-resettable seeds are embedded within the TPM. They are stored within TPM and are used as an input for key derivation functions for storing and loading keys. This ensures when this seed is reset, all previously managed keys are invalidated. A seed is used to create the parent key of other managed keys. A parent key will be needed to load the keys that are protected by that key. This makes it possible to grant another TPM access to keys by duplicating the parent key. Additionally, this means once the parent key is destroyed, all its child keys will become inaccessible. This makes it possible to quickly, selectively, and irreversibly destroy all keys during device reset. During the device reset, the seeds for the owner hierarchy will be reset so that all keys under the owner hierarchy will be invalided, while other hierarchies and keys used by other components will be kept accessible.

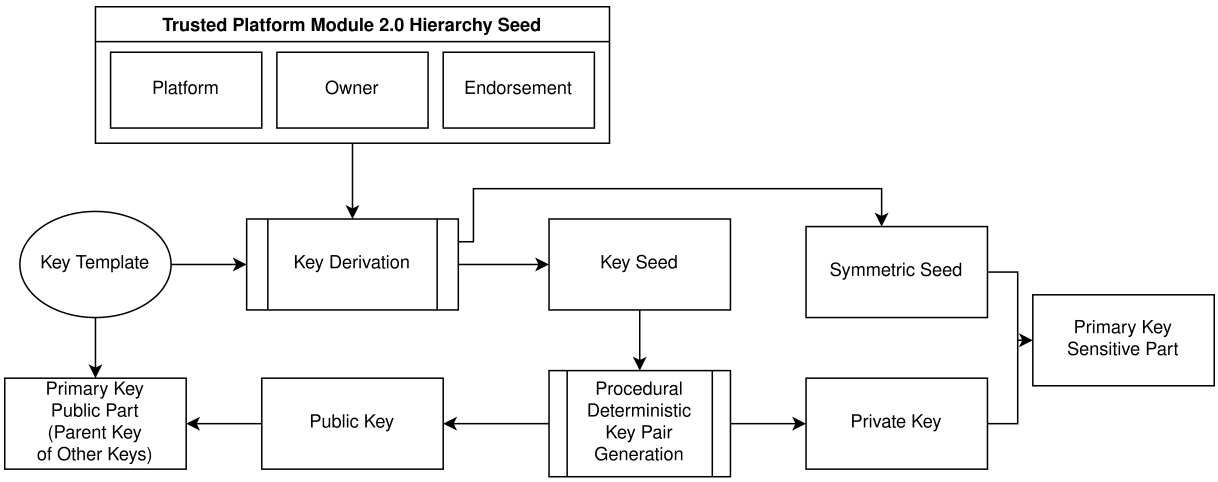

Figure 2.1: This figure shows the procedural of generating a primary key in Trusted Platform Module 2.0. The key derivation process takes a key template and one of the resettable seeds and outputs the data to procedurally generate the private key and other sensitive data.

TPM usually supports 3 kinds of keys: HMAC key, Asymmetric key, and Symmetric key. HMAC key and Asymmetric key can be used to create a tree of managed keys. For different kinds of keys, different operations can be used. For Symmetric keys, encryption and decryption can be used. HMAC keys, Asymmetric keys can be used to sign, verify

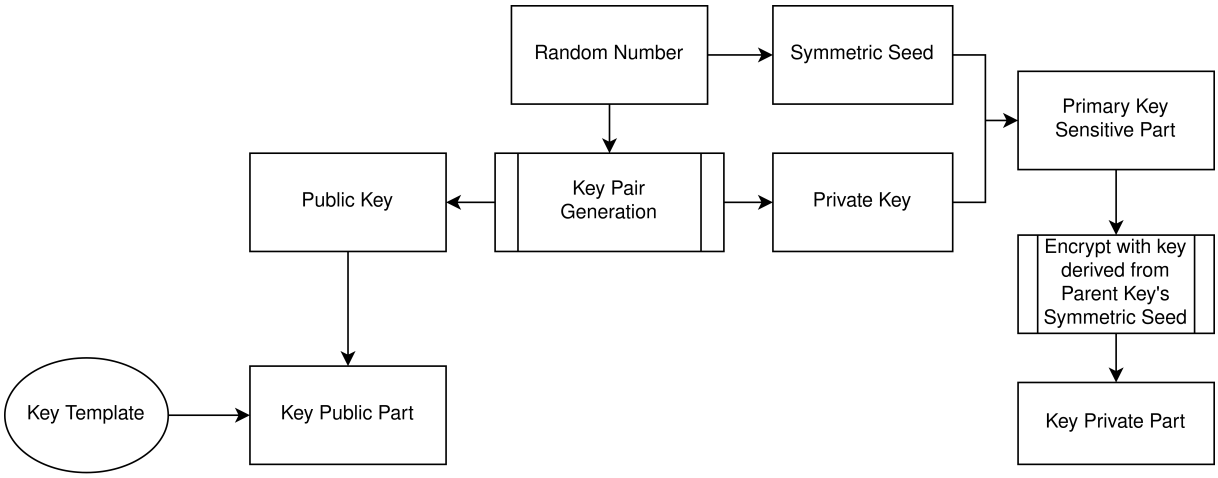

Figure 2.2: This figure shows the procedural of generating a key in Trusted Platform Module 2.0. The private key will be randomly generated(unless requested otherwise). The private part of the key will be encrypted with a key derived from the parent key's symmetric seed.

content depending on the key usage restriction. Each key can have properties associated with it. These properties include key usage restriction, key duplication restriction and key usage restriction. These properties allow users to construct the restriction of the key that will be enforced by TPM (Trusted Computing Group (2016)) . These keys can additionally have access control mechanisms. These managed keys can be used to complete cryptology operations without revealing the content of the secret key so that the keys can remain protected while retaining functionalities. This allows TLS certificates, GPG keys and other user credentials to be managed by TPM.

In addition to the keys, the TPM also manages non-volatile storage. There are several types of non-volatile storage. In addition to the ordinary buffer, bitmap, counter, extend, and pin pass and pin fail are also supported. These special types of buffer allow a restricted set of interactions on the buffer for writing. For bitmap, only setting individual bit is allowed, once set a bit cannot be cleared. For counters, only increasing counters are allowed. This allows more complex logic to be created by combining this feature with other settings to create an individually revocable key or other more complex restrictions. The non-volatile storage within the TPM can only be accessed with commands, and cannot be extracted from the device. This can be used to prevent the recovery of encryption metadata from deleted storage that could allow an attacker to decrypt content with the previous version of the password.

In addition to the parent key, there can be additional access control methods. A password can be used to protect the content managed by TPM. Compared to software-based key-derivation encryption schemes, an anti-dictionary attack mechanism is built-in

into TPM that allows a burstable password input attempt while preventing attackers from mounting a distributed attack on a computer cluster or continuous attempts. Additionally, the authentication can be completed with a policy-based authentication. A policy can be based on a password, a non-volatile storage value that satisfies a specific requirement, the internal status of TPM like clock, the specific condition about the operation such as its command, the specific condition about the target such as the target non-volatile storage is not written, and/or a logical combination of all above. This allows rather complex logic to be enforced on the TPM without the need to support them individually. Specifically, boolean operations on conditions are supported to allow developers to create applications that allow users to unlock the key for disk encryption or authentication with different passwords to mix different safety measures such as password, device status, or non-violate storage status to create a multi-factor protected credential.

For operating systems, it is also possible to use the platform configuration register to perform a measured boot. Keys can be restricted to be unlocked only when the system matches a predefined state. The state of the system is supplied by the system and kept in an append-only log within the TPM. This makes sure that if the system software is compromised, the content protected by TPM will be inaccessible to the attacker. This has allowed some full disk encryption software to unlock the hard drive automatically instead of inputting a password on every boot.

Interaction Interface

The interaction with TPM is accomplished with the Trusted Platform Module Library API. The TPM library will connect to the TPM running on the same system. For each programming language, a separate library is needed. Since so many functions are defined in the API, usually only a subset of them is available in non-reference implementations.

In order to support TPM, it will need to be supported at various levels of the system. This includes a hardware module, motherboard/SoC support, kernel support, and user mode library support. In order to share TPM between different applications, a resource manager will be needed. On both Linux and Windows, such a resource manager is provided by the system. Since the TPM standard is rather complicated, this kind of support takes significant time to be completed, for example, the TPM 2.0 standard is published in 2013(Trusted Computing Group (2013)) , and the support for Resource Manager introduced in Linux Kernel in 2017 (Jarkko Sakkinen (2017)) (James Bottomley (2017)) , while TPM is yet to receive significant adoption from software other than operating systems. The reference implementation of TPM and its associated tools supports the majority of the functions provided by TPM. However, third party libraries usually only

support a subset of functionality.

Typical Application Design

An Application using TPM is usually independent of the manufacturer of TPM. It will use the TPM module on the user's device. The application will use the API provided to create and operate on resources managed by the system. For operating systems, the system status can be used to define policies that restrict the use of keys to a certain environment with platform configuration registers. This facility is not available to ordinary programs since ordinary programs cannot control or interpret the value of platform configuration registers. For ordinary programs, TPM can be used as a security co-processor that holds an unextractable key. This is usually created to protect against dictionary attacks so that the attackers are rate limited by the TPM module with anti-dictionary attack setting and the speed limitation of the TPM module. Regardless of the type of application, it will need to handle the situation that a TPM lacks some functionality since it is not required for the TPM to support all operations.

2.3.3 Trusted Execution Environment

The Trusted Execution Environment is typically used in smartphones and other ARM-based devices (Li et al. (2019)) . It is originally defined by the Open Mobile Terminal Platform as a set of software and hardware to protect sensitive information from both software and hardware attacks (Open Mobile Terminal Platform (2009)) .

The Trusted Execution Environment is programmable and has sufficient resources for most tasks. It, however, still suffers from the same one vendor per device limitation.

On Android devices, the Trusted Execution Environment is used to create an Android KeyStore that stores credentials and allows applications to store encryption keys, and Android KeyGuard limits password input rates to protect users from dictionary attacks. On other devices, the Trusted Execution Environment environment can also be used to create hardware wallets that understand cryptocurrency signature schemes to protect users' financial assets.

Overall Design

The Trusted Execution Environment is an isolated region within the ARM System on Chip (Hua et al. (2017)) (Mukhtar et al. (2019)) . It has exclusive access to resources designated to it, and special access to some of the IO pins associated with it. In Trusted Execution Environment the System on Chip keeps two isolated execution environments (Jacomme et al. (2017)) , an ordinary execution environment(Rich Execution Environment), and a

Trusted Execution Environment. Both of them are running on the same chip but separated into different execution environments by the System on Chip processor. They will have separate kernels and userspace. The communication between these two separate instances is conducted through secure world monitor's switching between these two worlds. (Hua et al. (2017))

The Trusted Execution Environment has full access to the ordinary execution environment's resources. This allows it to access the resources referenced in the invoking command. The kernel and applications running in ordinary execution environments do not have access to the memory space or io device owned by the secure execution environment. This allows two environments to cooperate with each other and share critical information with each other. The image of the Trusted Execution Environment is signed with a key that is programmed into the chip itself. This prevents the loading of unrecognized images. For commercially available devices, this pinned public key is programmed in the factory, and the signing key is not available for the end-user. This, however, makes it infeasible for third party developers to utilize this facility for their own needs.

The Trusted Execution Environment has access to a unique key embedded within the chip that can be used as the master key for other operations. In addition to that, a true random number generator, and a secure clock is typically provided to the Trusted Execution Environment environment. All of them make it harder to influence the behaviour of the Trusted Execution Environment from the ordinary execution environment. The key is inaccessible to the rich execution environment and allows the Trusted Execution Environment to derive a key that is inaccessible to the rich execution environment.

Since the Trusted Execution Environment may have exclusive access to IO devices connected to the chip, it is possible to take over the access of specific peripherals and prevent the software running within the ordinary execution environment from modifying the output content of the Trusted Execution Environment or receive input from the user.

Programming Interface

Trusted Execution Environment has a separate SDK to develop user-mode software and kernel-mode operating systems. The Trusted Execution Environment usually has an operating system, user model software can use the API provided by the respective operating system running inside the Trusted Execution Environment. Applications running in Trusted Execution Environment usually have 2 - 8 MBytes of volatile memory and can access the same storage as the application running in an ordinary execution environment.

The libraries designed for the C language can be used in creating the Trusted Execution Environment since Trusted Execution Environment has a very similar execution environ-

ment as the rich execution environment; most libraries for embedded environments will retain functionality. This makes it possible to reuse the diverse library ecosystem available for embedded firmware development.

The operating system within the Trusted Execution Environment also provides significant support to the user space applications running within the Trusted Execution Environment environment. This includes a secure file system that stores the content of the file in either an ordinary file system or Replay Protected Memory Block(Reddy et al. (2015)) . When a Replay Protected Memory Block is used, the content stored within it will be protected from reading, modification, deletion. When an ordinary filesystem is used, the content will be protected from reading. If the metadata of the file is stored in the Replay Protected Memory Block and the content is stored in an ordinary file system, the content will be protected from reading and modification. These varying degrees of protection can be used by the application to provide a sufficient amount of protection for data while being conservative about resources when possible.

A application running within the Trusted Execution Environment can be either shipped within the Trusted Execution Environment operating system itself or be loaded from an ordinary file system. The application loaded from an ordinary file system will not be vulnerable to downgrade attack since both the content and version are signed, and a version database will be maintained within the Trusted Execution Environment environment.

Interaction Interface

Applications running in an ordinary execution environment can invoke applications running in a Trusted Execution Environment with a trap instruction. This interaction is similar to system calls to the kernel running in the same operating system. The application running in Trusted Execution Environment has unrestricted access to the memory held in the ordinary execution environment and can follow the address in the system call.

The client API contains only two parts: session management and command invocation. This allows clients to reduce the amount of logic and make it easier to create a comprehensive API.

2.3.4 Typical Application Design

In a typical Trusted Execution Environment based application, the vendor will usually need to be the manufacturer of the device or be approved by the manufacturer of the device. The application running within the Trusted Execution Environment will accept the command from the ordinary execution environment, checking it against constraints. If the constraints are met, the application running in Trusted Execution Environment will

complete the action with cryptology keys managed by Trusted Execution Environment and return the result to the trusted computing client. The application interacting with Trusted Execution Environment from the ordinary execution environment is typically created by the same vendor as the application running inside the Trusted Execution Environment.

2.4 Analysis

2.4.1 Present Day Trusted Computing Limitations

One firmware vendor per device

For all programmable trusted computing implementations discussed, only the manufacturer of the trusted computing device can approve a program that runs on the trusted computing facility. This restriction is usually enforced by requiring the program to be signed or loaded with a key held by the manufacturer.

Performance limitation

The performance of trusted computing devices can be slow. This is especially true for physical devices that are dedicated to trusted computing purposes since the cost of such a device cannot be amortized by reusing the device for other purposes.

Functionality limitation

The functionality of trusted computing devices is usually limited to the set of functionality defined by the manufacturer. This means it is usually hard for a independent developer to utilize the trusted computing devices unless the application is specifically designed for that trusted computing device since there are various limitations around the API provided by the trusted computing device. There is a very limited amount of supported algorithms and methods of operation. The program has to be designed to work with these trusted computing platforms in the beginning in order to be supported by these devices.

The same thing is true regardless of whether the device is a programmable trusted computing device or is a fixed-function trusted computing device. In either case, an unaffiliated party will not be able to customize logic to make the trusted computing system generally useful.

2.4.2 Cause of Limitations

Restriction Based Protection

Currently, all programmable trusted computing implementations discussed protects the integrity of the platform by making sure that only programs that are signed by the manufacturer can run on the device. This design is simple and intuitive. If only the programs authorized by the manufacturer can run on the device, then the resources on the device are exclusive to the manufacturer and therefore can be trusted by the manufacturer. This approach makes it more difficult to share the execution environment since the execution environment's access to resources is unrestricted.

Cost of One Firmware Vendor per Device

The restriction based protection makes it impossible to share the programmable trusted computing implementation with entities without an approval process from the device manufacturer. This means each vendor of a programmable trusted computing facility usually needs to give their customer their own physical trusted computing device. This cost cannot be shared by multiple vendors. This makes it very difficult to create a rich experience with trusted computing since the resource is usually limited.

2.5 Academic Research

Currently, commercially available implementation of trusted computing designs that have been just described cannot enable multiple uncoordinated vendors to use the same underlying hardware with fully programmable logic. There are several academic attempts to enable the sharing of trusted computing facilities to different vendors, however, none of them achieved all desired characteristics.

Dynamic Java Card Applet provision for payment applications still requires cooperation from device manufactures(Alimi and Pasquet (2009)) . This kind of approach does not allow the sharing of trusted computing faculty to uncoordinated trusted computing vendors, is not platform-neutral, and only supports payments applications.

SecTEE (Zhao et al. (2019)) attempts to share the same underlying trusted computing facility to different vendors with dynamic enclave loading mechanics. However, this approach is neither platform-neutral nor promotes the data protection rights of users.

Open-TEE with white-box cryptography (Bicakci et al. (2019)) attempts to create a software-only trusted computing environment with white-box cryptography (Chow et al. (2002)) . However, only a limited amount of cryptologic schemes are supported by white-

box cryptography, and the logic that enforces the usage limitation of these cryptography primitives cannot be protected with white-box cryptography.

The Trusted Execution Module (Costan et al. (2008)) attempts to create a general purpose trusted execution engine with an external embedded environment. It has a strict resource and performance limitation and does not promote user's data protection rights by design.

Trusted Platform Model Assisted Trusted Computing Hypervisor has been attempted (Zhang et al. (2007)) . However, this depends on hypervisor software to be unmodified, thus requires trusting the hypervisor operator and does not protect against an attacker with full physical access to the device.

2.6 Conclusion

The limitations of the currently available programmable trusted computing environments are created by the status quo of one vendor per device model. Since the device cannot be shared among different vendors, the cost becomes a significant factor that limits the performance and usability of the trusted computing device. The high barrier of entry that requires a physical device to be shipped to the user also limits the usefulness of the trusted computing technology.

The one vendor per device model is created by the current restriction based integrity protection. It verifies the executable code by restricting any codes from third-party developers from being loaded and executed in the trusted execution facility. This denied the attacker access to the trusted execution facility, but also prevented other vendors from reusing the device and sharing the cost of the physical device.

Non-programmable fixed-function trusted computing devices can be shared by different vendors, however, their utility value is limited by the semantic of provided functions.

In order to solve the issues discussed above, a general-purpose trusted computing platform needs to be both programmable and shareable between different vendors.

Chapter 3

Design

3.1 Overview of Approach

The Trusted Computing Platform is an isolated execution environment. The identity of the Trusted Computing Platform and the Trusted Computing firmware vendor is authenticated with Public Key Infrastructure. Each program has a signed manifest, which contains the expected method of interaction between a trusted computing platform and executable code. The executable code for the Trusted Computing firmware is represented in Web Assembly, which allows it to be platform-independent and be supported universally. Every input and output of the Trusted Computing firmware is managed by the trusted computing platform, and their restrictions are enforced by the Trusted Computing Platform, ensuring the protection of both vendor data and user data. Each Trusted Computing firmware have its own persistent state in the Trusted Computing Platform.

The Trusted Computing Platform isolated environment has its own memory, its own execution context, random number generator, and its own non-volatile memory. In addition to these internal components, it can also have direct access to peripherals such as screens, and buttons. These resources are utilized by the Trusted Computing Platform to complete calculations and input-output data without interference from the ordinary execution environment.

3.1.1 General Usage Pattern

The developers that target the Trusted Computing Platform will create a firmware to run their logic at the Trusted Computing Platform. Once installed at a certain Trusted Computing Platform, an instance of that firmware will be created. This instance can be invoked to run functionalities contained within the firmware. The input and outputs defined for this function invocation are managed by the Trusted Computing Platform.

Some inputs are stored within an encrypted container: Envelope that is only accessible to the Trusted Computing Platform so that the Trusted Computing Platform can verify the constraints associated with it before providing the input to the function. The functions within the firmware can also use the non-replayable non-volatile storage to create permanent and irreversible action. To securely communicate with the vendor's server or other Trusted Computing Platforms, a Session can be established to send encrypted containers: Envelope to the remote device.

3.1.2 Firmware

The programmable application running on a Trusted Computing Platform is called a Trusted Computing Firmware. Different firmware from different vendors can co-exist on the same device. Each firmware will have a manifest file containing all important information about it. This file will be considered the primary entry point of the application.

3.1.3 Envelope

In the Trusted Computing Platform, some inputs or outputs can be encrypted and authenticated. These encrypted contents are in an Envelope format interpretable by the Trusted Computing Platform.

When using an encrypted Envelope packet as input, the trusted computing client will provide hints about the Session identity, so that an appropriate Trusted Computing Platform managed key can be used to decrypt the content. If the authenticated decryption is successful, the plaintext content of the Envelope will be parsed. A similar process will be used to wrap the output if requested by the trusted computing client.

A packet's unencrypted content is as follows:

Name	**Type**	**Omissible**
restrictionsDigest	bytes	
content	bytes	OPTIONAL
padding	bytes	OPTIONAL

Table 3.1: A unencrypted version of envelope contains restrictions digest, content, and padding.

The restrictionsDigest field is the digest of the binary representation of restrictions applied to the input. The Trusted Computing Platform should use this value to verify the trusted computing client has satisfied the restriction of using this input. The value of restrictions is provided by the trusted computing client.

The content field is the content of the Envelope that will be used as the raw input that is provided to the firmware's functions.

The padding field is used to pad the length of the input or output. This can prevent the leak or extraction of sensitive information from the Trusted Computing Platform. This can make the output of firmware constant length so that a sensitive input can protect its information by restricting the vary time execution and vary length output. This makes sure that without the appropriate decryption key, the output of the function contains no information about the sensitive data for a party without the decryption key.

3.1.4 Functions

In the Trusted Computing Platform, Function invocation is the primary way for Trusted Computing Firmwares to interact with trusted computing clients from the ordinary execution environment.

The functions reference exported symbols from a Web Assembly executable binary. The Trusted Computing Platform will verify the executable binary against its record in the manifest before accepting it. The execution will be completed entirely in the Trusted Computing Platform to ensure the processing cannot be interfered by an attacker.

The inputs and outputs of the Functions will be defined by the trusted computing client. The input of the function will be checked by the Trusted Computing Platform against the restrictions defined in the manifest.

3.1.5 Non-Volatile Memory

In the Trusted Computing Platform, Non-Volatile Memory Storage is provided to both the Trusted Computing Platform itself and the firmware running on it. Non-Volatile Memory or at least a digest of it is stored in shielded storage or Replay Protected Memory Block (Reddy et al. (2015)) so that the content cannot be forged and is non-replayable.

Upon the installation of firmware, the non-volatile memory for the firmware will be initialized. Each version of the same firmware will have separate entries. However, Functions can access storage in an earlier version of the firmware to facilitate the transfer of updates.

3.1.6 Firmware Instance Data

In the Trusted Computing Platform, installing a firmware creates an instance of that firmware. This allows the trusted instance to keep a record of all the instances of firmwares within the replay protected storage. This makes sure that all operations performed on

this instance can be permanent and irreversible since all operations can be checked against the non-replayable record.

A Firmware Instance's status is as follows:

Name	Type	Omissible
installationNonce	bytes	
manifestDigest	bytes	
firmwareNonVolatileMemoryDigest	bytes	

Table 3.2: A firmware instance data contains installation nonce, manifest digest and firmware non-volatile memory digest.

The installationNonce field is randomly generated on the first installation of the firmware. This can be used to install several instances of the same firmware on the same device. Additionally, this ensures once uninstalled, the Envelopes associated with that instance of the firmware will be invalided.

The manifestDigest field is the digest of the manifest of the firmware. The signature of this value is verified on the installation. By storing this information within the non-volatile memory, the value of manifest cannot be forged.

The firmwareNonVolatileMemoryDigest field is the digest of a non-volatile memory of the firmware in the format of NonVolatileMemoryRecords. By combining all non-volatile memory entries into a single field, the update to the content is atomic, and the space requirement is minimal and consistent. This prevents the attacker from forcing the Trusted Computing Platform into making partial updates of non-volatile memory, by making sure partially updated content is invalid. Also, this enables the firmware developer to use a significant amount of non-volatile storage without negatively influencing the operation of other firmware.

The original data of NonVolatileMemoryRecords is stored in a trusted computing client, but encrypted and authenticated with a key only available to the Trusted Computing Platform. The user of firmware needs to submit the wrapped data on invocation, and retrieve an updated version of the wrapped data.

A NonVolatileMemoryRecords is as follows:

Name	Type	Omissible
records	array of NonVolatileMemoryRecord	OPTIONAL

Table 3.3: A non-volatile memory records contains a array of NonVolatileMemoryRecord.

A NonVolatileMemoryRecord is as follows:

Name	Type	Omissible
nameDigest	bytes	
content	bytes	OPTIONAL
padding	bytes	OPTIONAL

Table 3.4: A non-volatile memory record contains the digest of non-volatile record, content of the record, and a padding.

The nameDigest field is the digest of the non-volatile memory name as it appears in the manifest. The name is a digest to ensure the length is predictable.

The content field is the content of the non-volatile memory.

The padding field is the padding. This ensures the content is always a constant length. This prevents any leak of information with varying length data. This is necessary since the content of the non-volatile storage will be swapped out to the trusted computing client. The length information is otherwise exposed.

3.1.7 Session

In the Trusted Computing Platform, the Session is the primary way for two instances of Trusted Computing Firmware to send encrypted and authenticated data with each other. A Session is established between two Trusted Computing Platforms, or a Trusted Computing Platform and a vendor. The Session status is stored in the client in an encrypted and authenticated form.

A SessionStatus is as follows:

Name	Type	Omissible
sessionKey	bytes	
sessionRoleAcceptor	boolean	
sessionPeerDigest	bytes	

Table 3.5: A unencrypted version of session status record contains the session key, the role of the communication, and the digest of session peer certificate.

The sessionKey field is the encryption key. This value allows the Trusted Computing Platform to send encrypted messages to the other end of the session and decrypt the content if requested.

The sessionRoleAcceptor field is the role in the session. This value allows the Trusted Computing Platform to use different keys for content sending in different directions.

The sessionPeerDigest field is the peer's information. This allows the Trusted Computing Platform to enforce restrictions based on the properties of the remote peer. This

also allows the Trusted Computing Platform to display authenticated peer information to the user when necessary.

3.2 Public Key Infrastructure

Public key infrastructure in Trusted Computing Platform has two roles, the verification of the Trusted Computing Platform devices, and the verification of Trusted Computing Platform firmware vendors. For this reason, two separate kinds of public key infrastructure hierarchies are used. For each hierarchy, each entity in the hierarchy is represented as an X.509(D. Cooper and et al (2008)) format certificate.

3.2.1 Device Hierarchy

Each Trusted Computing Platform manufacturer has a self-signed root certificate for device attestation. Every Trusted Computing Platform has a unique attestation certificate embedded in the device issued by the Trusted Computing Platform manufacturer issued by Trusted Computing Platform manufacturer's root certificate. This certificate is imported into the device in the factory and cannot be extracted from the device. Other Trusted Computing Platform devices and Trusted Computing Platform firmware vendors can use this to verify that the communication peer is a Trusted Computing Platform. This design is similar to that of Android Device Attestation(Android Open Source Project and et al (2020b)) .

3.2.2 Firmware Vendors Hierarchy

Each Trusted Computing Platform firmware is signed by a vendor. Each update of the firmware is required to be signed with a certificate issued by the same self-signed Firmware Vendors root certificate. However, any Firmware Vendors root certificate will be accepted by the Trusted Computing Platform on the first installation, which means everyone can be a vendor. Different firmware and vendor certificates coexist with each other. This design is similar to that of Android Application Signing(Android Open Source Project and et al (2020a)) . This information also authenticates remote servers to create Sessions.

Firmware Vendor root certificates can be used to issue additional certificates to sign firmware updates. During the install or update of firmware, the trusted computing client will submit the certificate chain. The Trusted Computing Platform will verify the certificate chain before accepting it as an update of an existing application.

3.3 Signed Program Manifest

Each firmware has a manifest associated with it, containing the definition of functions, restrictions, and the resources required by the firmware. The manifest can be interpreted by the Trusted Computing Platform and its software development kit. The definitions and restrictions within manifest will be enforced by the Trusted Computing Platform itself, which cannot be influenced by software in the ordinary execution environment.

In the following paragraph, the information stored in the Signed Program Manifest is described.

3.3.1 Firmware vendor

The firmware vendor information is designed to facilitate the update of applications and authenticated communication with the vendor. It is represented as a digest of the Firmware Vendor root certificate.

The Trusted Computing Platform will use this information to construct the Firmware Vendors Hierarchy Public Key Infrastructure. During an install or update of an application, the certificate chain of firmware including the root certificate will be provided by the trusted computing client. The root certificate will be checked against the hash stored within the manifest, other certificates will be checked based on the signature of its issuer.

3.3.2 Firmware Identifier

The Firmware Identifier information includes the name and version of the firmware. The firmware name allows the software vendor to create multiple firmware without security complications. This prevents a trusted computing client from updating firmware with another firmware created by the same vendor.

The firmware version allows the firmware to be updated and facilitate the ratchet of state.

3.3.3 Function Declaration

The Firmware Function Declaration information includes the name of the functions being declared, their input, output and the minimal restrictions applied to the function, input and outputs.

Only declared functions will be available for invocation on the Trusted Computing Platform firmware. Before any invocation, the conditions of the input and output need

to be specified by the trusted computing client. The restriction must be at least as strict as those in the manifest.

The restrictions defined in the manifest are mandatory restrictions. The invocation will not be successful unless all constraints in the functions declaration exist at the function invocation. This ensures the software vendor can define the minimal restrictions that protect the sensitive information of the vendor.

3.3.4 Non-Volatile storage Declaration

The firmware non-volatile storage declaration information includes the persistent storage of the program. This includes the name, and size of persistent storage required.

All the non-volatile storage required by the trusted computing firmware will be declared here. This allows the size of storage to be know at the installation time, thus allowing the Trusted Computing Platform to always pad the storage to the maximum size.

3.3.5 Firmware

The firmware information includes a digest of the executable firmware binary. This information allows the Trusted Computing Platform to verify the executable firmware binary.

In any function invocation, the invocation request will contain the firmware binary. Before this firmware binary is executed, the digest value of this firmware binary will be checked against the record in the firmware. This allows the Trusted Computing Platform to dynamically accept different executable binary based on the manifest without the need to store any executable binary within its own persistent memory.

3.4 Platform Agnostic Firmware

Firmware is required to be portable and vendor agnostic.

The Web Assembly(Haas et al. (2017)) standard is chosen as the standard representation of executable code for a few reasons: It has existing toolchains, It does not assume a predefined application binary interface and is hardware neutral, It has already been used in browsers and by cryptocurrencies smart contracts. All these characteristics mean Web Assembly binaries is conservative in size, can be developed with existing tools that do not have incorrect assumptions, and there are relatively few issues with the implementation of the library associated with Web Assembly.

Toolchains for other instruction set architectures often have assumptions about the system. This includes the existence of an operating system. This is not true for a Trusted

Computing Platform as the trusted computing firmware is running within the Trusted Computing Platform sandbox, without access to a file system or other operating system services.

The execution engine will handle the input and output of the Web Assembly program. The Web Assembly execution environment will be discarded after a single use.

3.4.1 Input and Output Format

Both input and output are required to be encoded as Concise Binary Object Representation(C. Bormann et al. (2020)) binary data. The execution environment will invoke buffer manipulation functions to set the input and output data.

3.4.2 Execution Environment

The Web Assembly program runs in an isolated environment and has no access to any external components. This ensures all the input and output will be subject to the limit set out by the manifest and application.

3.5 Protected and Restricted Input & Output

The input and outputs of functions will be managed by the Trusted Computing Platform. The trusted computing client will define the inputs and outputs, and provide the necessary information associated with them. The Trusted Computing Platform will check if the conditions are met before providing the supplied information to the Platform Agnostic Firmware executable binary. Some of the input is wrapped in Packet format and only the Trusted Computing Platform has the key to unwrap the contents contained.

This design can be used to protect both the user and firmware vendor. Firmware vendors can create restricted input and output to make sure that sensitive information is only available to the vendor. Users can have special inputs that have restrictions on its use, making it inaccessible to the software vendor while retaining the functionality of the program.

3.5.1 Types

There are 5 kinds of input and outputs, Plaintext, Protected Input and Output, Storage, Random, and Session Peer.

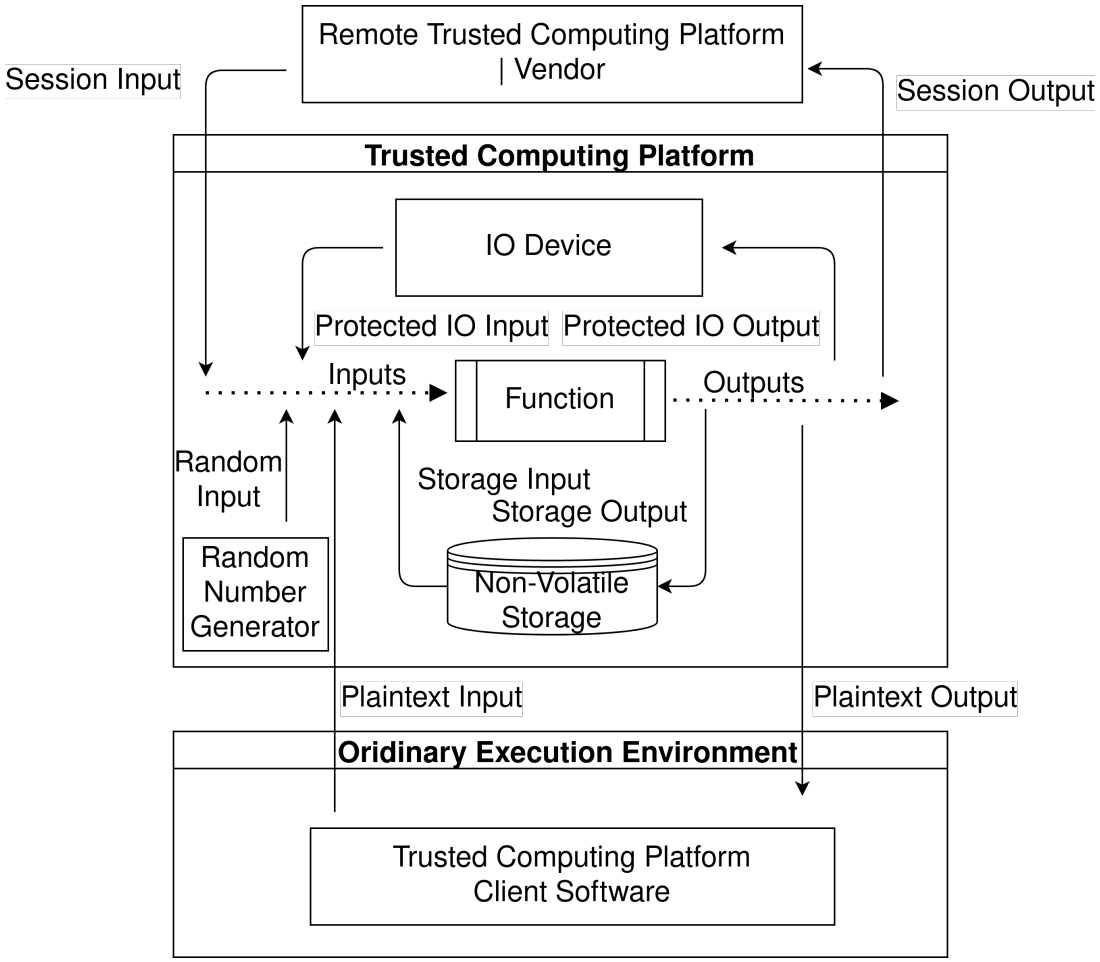

Figure 3.1: This figure shows the flow of data during a function invocation. The contents of inputs and outputs are managed by the Trusted Computing Platform.

Plaintext

The trusted computing firmware can have plain text input and output. This is typically the way the trusted firmware interacts with the trusted computing client.

Random

The trusted computing firmware can have random input. Random is required for the correct operation of many cryptology primitives.

This random value should be generated from a true random number generator within the Trusted Computing Platform.

Storage

The trusted computing firmware can have non-volatile storage input and output. Storage input and output need to be specified with the version of the program.

If a non-volatile storage input is specified, that data will be read by the Trusted Computing Platform and supplied to the firmware as input. If a non-volatile storage output is specified, that output will be written to the non-volatile storage specified. Each non-volatile storage can be used only once, and unless the same value is copied to the output with the same name, the old value will become invalid.

Protected Input and Output

The trusted computing firmware can have Protected I/O input and output.

The Trusted Computing Platform can collect protected input from users directly when requested by the trusted computing client. The input collection request includes the type of input being collected and the dynamic mandatory restrictions that will be applied to the input. During the input collection, the Trusted Computing Platform will obtain exclusive access to the input and output devices like touch screens, and show the dynamic mandatory restrictions applied to the data before accepting input from users. The protected input or output use a container format that includes metadata about the data including dynamic mandatory restrictions associated with this data. This allows the input data to have additional restrictions on its usage, protecting the information inputted by the user. The protected output from the trusted firmware is encrypted with a key hold by the Trusted Computing Platform and its content can be displayed by the Trusted Computing Platform to screen directly when requested by a trusted computing client. This allows sensitive output to be displayed to the user without making it accessible to the potentially compromised system.

An input or output of Protected Input and Output can have dynamic mandatory restrictions associated with it. These dynamic mandatory restrictions are stored within the data packet container format and cannot be modified while avoiding detection. These dynamic mandatory restrictions is then subsequently applied to the input.

Session Peer

The trusted computing firmware can have Session Peer input and output.

The Trusted Computing Platform can create encrypted sessions with remote peers such as other Trusted Computing Platforms, vendor's server, or local peers such as itself.

The encrypted messages can only be decrypted at the other end of the session.

Session input or output can have dynamic mandatory restrictions associated with it in a way similar to protected input and output.

3.6 Restrictions

The input and output data can have different types of restrictions attached to it in addition to the type.

3.6.1 Type Restriction

An input or output can have a type restriction applied to it. In this restriction, the expected type of this input or output can be defined. If the actual type of the input or output does not match the expected value, this restriction will prevent the invocation from being completed successfully. This restriction is designed to be used on inputs that expects plain text or random inputs, or outputs that generates plain text outputs. For inputs that expect random number inputs, this restriction prevents an attacker from supplying non-random number data to attack the firmware, as some primitives require high-quality random numbers (Johnson et al. (2001)) . For plain text inputs or outputs, this restriction prevents such input or output to be used for extracting or overriding other types of inputs.

3.6.2 Session Restriction

An input or output can have a session restriction applied to it. A session restriction defines the communication peer of a session. It can be a session nonce, the unique identifier of the session, a peer certificate hash, a flag set that assert the type of the certificate, or any combination of these restrictions. If the type of input or output is not a session or any of the session detail does not match, this restriction will prevent the invocation from being completed successfully.

Session restriction can prevent input from receive data from receiving unrecognized peer, or output from sending data to an unintended recipient.

3.6.3 Non-Volatile Storage Restriction

An input or output can have a non-volatile storage restriction applied to it. A session restriction defines the non-volatile storage that can be used as the input or output. A non-volatile storage restriction can contain the name and the version of the non-volatile storage. The name restriction allows the storage slot of the non-volatile storage to be

defined, to prevent an unexpected storage slot from being used as input or output. The version restriction allows the non-volatile storage to access a different version's instance of the non-volatile storage, thus make it possible to upgrade the status of a previous installation.

3.6.4 Output Minimal Constraint Restriction

An input can have an output minimal constraint restriction applied to it. It can contain a set of other instructions. If such a constraint is found on input, then all outputs in the same invocation must have all the restrictions specified in this restriction. This ensures any outputs that could contain an input's information are subject to the restrictions defined by the input.

3.6.5 Instance Vendor Restriction

An input can have an instance vendor restriction applied to it. It can contain the firmware's nonce, vendor certificate digest, and version. If a restriction is specified, the firmware to be invoked must much the restriction. Otherwise, the invocation will fail. This can be used to make sure that a session packet or protected input can only be used by a certain installation of an app, a vendor's app, and/or a specific version of the app. This makes it possible to prevent other firmware installed on the same Trusted Computing Platform from access data not intended for them.

3.6.6 Input Output Name Restriction

An input can have an input-output name restriction applied to it. An input-output name restriction can have a pinned input name, and function name within it. It is required that the input that uses this input must match the input name and function name. Otherwise, the invocation will fail. This can be used to make sure that the input is being used for the intended purpose, not misused as another input.

3.6.7 Or Restriction

An input or output can have an or restriction applied to it. An or restriction contains a set of other restrictions. An or restriction is considered satisfied if more than one of the restrictions contained in this or restriction is satisfied. This can be used to allow the usage of input in more than one situation.

3.6.8 And Restriction

An input or output can have an and restriction applied to it. An and restriction contains a set of other restrictions. An and restriction is considered satisfied if all restrictions contained in this and restriction is satisfied. This can be used with or restriction to make sure that a set of restrictions have to be satisfied before an or restriction is satisfied.

3.6.9 Constant Time Restriction

The input can require the function to run at a constant time. This can prevent the leak of timing information. If the actual execution time is shorter than the time specified in this restriction, then the output will not be returned to the trusted computing client until the specified amount of time is passed. If the execution is not finished when the time specified in this restriction has passed, then the output content will be empty. In either case, the output data will be padded to prevent the leak of data length.

3.7 Protected Peer Data Exchange

In order to facilitate the secure exchange of information, The Trusted Computing Platform can establish Session with remote peers.

The establishment of the Session is facilitated with a standard TLS mutually authenticated connection. Both local and remote will present a certificate to the remote peer. Depending on the expectation of the Session, different sets of certificates will be accepted. Regardless of the certificate accepted, the details of the session will be recorded system-wide. The encryption key associated with the session will be recorded as well. These recorded data about the certificate can subsequently be used to enforce restrictions on session inputs or outputs. By delaying the enforcement of certificate requirements to the function invocation, a more precise and fine-grained restriction can be used.

3.8 API

The interaction with the Trusted Computing Platform is facilitated through a set of API primitives.

3.8.1 Install Manifest

The trusted computing client application can request a manifest to be installed on the Trusted Computing Platform. The trusted computing client application provides the

manifest and its signature to the Trusted Computing Platform. The Trusted Computing Platform validates the content, allocates the necessary resources and creates the status for the firmware.

3.8.2 Run Function

The trusted computing client application can request a function to be run on the Trusted Computing Platform. The trusted computing client application needs to provide a Manifest, Web Assembly binary firmware, inputs and outputs and their restriction. The Trusted Computing Platform validates that the firmware's content matches the digest declared in the manifest, the input and output are valid and their restrictions are met. Before executing the function, and return the output to the trusted computing client.

3.8.3 Request Protected Input

The trusted computing client application can request a protected input. The trusted computing client application needs to provide the type of input required and the additional restriction to the output. The Trusted Computing Platform will show additional restrictions to the user before collecting user input. The result of user input will be provided to the invoking application in an encrypted form, available to be used as input to the trusted computing client.

During the execution of protected input requests, the Trusted Computing Platform will temporarily acquire exclusive control of input and output devices like touch screens. This ensures even if the attacker gained full access to the ordinary execution environment, the attacker cannot interfere with the protected input and output function of a Trusted Computing Platform.

3.8.4 Create/Accept Vendor/Peer Session

The trusted computing client application can request a session to be established. It is done by requesting the trusted computing client to relay a connection between two peers. The negotiation between two peers is done automatically by the Trusted Computing Platform. The session handle is then returned to the trusted computing client.

Chapter 4

Implementation

4.1 Overview

In order to demonstrate the general workflow and capacity of the Trusted Computing Platform design, a proof of concept implementation of this specification is created. It demonstrated the basic workflow of a Trusted Computing Platform based on the specification discussed earlier. It is designed to closely match the specification in order to showcase the typical workflow of a Trusted Computing Platform.

The program is written in Go language with standard go language libraries. This has reduced the dependency on external libraries. Go language is a general-purpose programming language with a balanced development speed and execution speed.

4.2 Public Key Infrastructure

The public key infrastructure in a Trusted Computing Platform uses the standard x509 public key infrastructure system with custom rules for the certificate authority.

The certificates are used as client and server certificates when creating peer sessions. The digest of peer certificates is stored in order to associate the session with the certificate.

The install/update certificates are issued with the certificate authority certificate pinned in the manifest file. The install/update certificate for each firmware has an extension with the digest of the manifest file. This certificate will be verified on application installation.

4.3 Manifest Interpreter

Each firmware has a manifest that contains the data described in the design sections. In the Proof of Concept Implementation, this value is encoded as a protocol buffer binary. Protocol Buffer(Kaur and Fuad (2010)) serialize structured data into its binary representation.

Since a protocol buffer formatted data is not human readable, protojson(Google and et al (2020c)) is used as a text representation of the manifest protobuf file. Protojson is the standardized method to encode a protocol buffer into JavaScript Object Notation format. JavaScript Object Notation (Nurseitov et al. (2009)) is a standardized method to encode unstructured data into a text representation resembling a subset of JavaScript.

During the installation of a firmware, the manifest and its signature will be verified. This ensures only a valid signed manifest will be accepted. However, as described earlier, anyone can create their own signing key. The update of the firmware requires a signing key issued by the root certificate specified in the current version of the manifest. This protected the information stored by the firmware from other firmware vendors while allowing anyone to deploy firmware on user devices without approval from the device manufacturer. Since the content of the manifest will also be checked, any errors in the manifest will be discovered in the installation phase.

4.4 Web Assembly Runtime

The executable code for firmware is represented as a Web Assembly binary. As described earlier, Web Assembly is the web standard for intermediary representation of executable code. The runtime in the Proof of Concept executes Web Assembly binaries with the Wasmer (Syrus Akbary and et al (2020)) Web Assembly execution engine. It is both fast enough and has high-level APIs that make creating applications significantly easier.

In the Proof of Concept Implementation, each Web Assembly binary will receive the input data in Concise Binary Object Representation format and generate output data in the form of Concise Binary Object Representation format. Concise Binary Object Representation is a binary representation of unstructured data. Data encoded in this format is both concise and self-explanatory. Unlike protocol buffers, it allows its data structure to be dynamically defined, which is required for the trusted execution platform to generate this data structure based on the input and output parameters for each function. Unlike JavaScript Object Notation, it represents data in a binary format that requires less conversion and overhead, which is important since the virtual machine for Web Assembly codes has limited resources.

4.5 Restricted Input & Output

Every input in the function invocation can have restrictions applied to it. In the Proof of Concept Implementation, the restrictions will be checked before invocation begins.

The checking of input and output is completed in 3 stages. Firstly, the input and output restrictions are checked against the manifest to ensure all restrictions are at least as strict as the manifest. Then every session packet or protected input's mandatory restrictions are checked against the restriction in the invocation to make sure that the mandatory restrictions associated with data are enforced. This enforces session restrictions as set by the remote session peer or protected input. Finally, every individual restriction on inputs and outputs are checked to make sure all the restrictions in the invocation requests are respected.

4.6 Protected Peer Data Exchange

Protected Peer Data Exchange is a two-phase process. In the first phase, two Trusted Computing Platforms negotiate an encryption key for future data exchanges. In the second phase, invocation output will be encrypted to the remote peer with the encryption key.

For the first phase, two Trusted Computing Platforms will communicate with TLS 1.3 protocol. Each side will present its own certificate as described in the public key infrastructure section. Trusted computing platforms will remember the peer certificate for future reference.

In the second phase, devices can send encrypted containers to each other. For each message, the container will be encrypted with the encryption key negotiated in the first phase. The encryption itself will use standardized authenticated encryption with associated data encryption(Bellare et al. (2003)) . The encryption key for different directions is different. This ensures the content sent from one Trusted Computing Platform cannot be used as input on that platform, and only the recipient can decrypt the content as input.

4.7 gRPC Interface

API is the primary way for the user to interact with the Trusted Computing Platform. In the Proof of Concept Implementation, the API is provided as gRPC API. gRPC(Google and et al (2020b)) is a standardized format for remote procedure calls based on gRPC.

The gRPC interface contains the simplified call for all APIs provided in the Trusted Computing Platform. Specifically, it allows the application to define a filename as input

data. This makes debugging and demonstration much easier, while not suitable for actual deployment to prevent privilege escalation.

4.8 Demo Application

To demonstrate the functionality of this proof of concept implementation, a demonstration application is created. In this demo application, a secure messaging functionality is implemented with this Trusted Computing Platform proof of concept implementation.

4.8.1 Background

Secure messaging applications protect user messages with encryption. Software-based secure messaging applications can protect information when the data is transferred through networks. A subset of them also claims to support peer to peer encryption, implying that the server operator does not retain access to the user's message. However, these claims cannot be independently verified by the user. Software-based secure messaging applications can share the encryption key to a third party, or share the encryption key with the server without being detected.

4.8.2 Design

The secure message application is split into 2 parts, the message sender and the message receiver. Both of them is are instances of this demo application, however, installed on different Trusted Computing Platforms.

Both Trusted Computing Platforms will establish a session first. This session will be used to transfer sensitive user data between these two Trusted Computing Platforms.

The sender application first requests a protected input with restrictions that limits its usage of send to a designated communication peer for display. This protected input is then transformed by the Trusted Computing Platform through send application firmware. An encrypted session packet is generated as output. This packet is transferred to the receiver application over a potentially compromised channel.

The receiver application will then receive an encrypted session packet. The receiver application will respect the restrictions specified in the encrypted session packet and display this content to the console directly, without returning the payload data to the receiver application.

4.8.3 Executable Web Assembly Firmware

The Web Assembly Firmware is created with the Rust language toolchain. With Rust's support for Web Assembly as a target platform, it is possible to create Web Assembly firmware without significant modification, provided that the application binary interface between the Trusted Computing Platform and firmware is WebAssembly System Interface.

The logic within firmware is a simple transformation of input data with string format before generating it as the output. This is not a significant amount of processing, yet it still demonstrated the possibility to support arbitrary data transformation logic within the trusted computing firmware.

4.8.4 Analysis

This demo application is a demonstration of the ability of the Trusted Computing Platform to protect the data protection rights of users. The user is able to check the restriction applied to their input data before inputting any data. Even if the sender device is compromised, it cannot evade the limitation that has been shown to the user. If the attacker changes the restriction during input collection, the user will be able to spot the difference in restriction, thus discovering the attack. If the attacker did not change the requested restrictions, the attacker would not be able to obtain the unencrypted version of the input or the sensitive user data contained within it.

Chapter 5

Evaluation

5.1 Use Cases

A trusted computing platform based on this specification has an infinite amount of possible use cases. However, different types of use cases will be analysed to understand the capability of this specification. In this section, the key difference to previously discussed multi-tenant trusted computing solutions will also be discussed.

Unextractable Key Storage

Unextractable Key Storage is a solution that allows a cryptology key to be used for its intended usage without allowing the key itself to be extracted from the device. This allows the cryptologic material to be unduplicatable, thus making it significantly harder to retain access to it without being discovered. There are a significant number of trusted computing solutions that support this use case including Trusted Platform Module 2.0.

With a fully programmable trusted computing platform based on this specification, significant amounts of the keys can be stored on a single trusted computing platform since the non-volatile storage consumes a constant size of protected storage. Comparing to smart cards that can only store 3-6 private keys on its non-volatile memory, support for holding significant amounts of key allow more applications to utilize the same trusted computing hardware without interfering with each other. The usage restrictions for the keys can be adjusted with programmable logic. The algorithms can be implemented in the programmable logic, thus, there is no limitation on the type of algorithms that can be supported by the firmware on the trusted computing platform. With peer session mechanics, the key materials can be transported from one trusted computing platform to another.

Trusted Platform Module 2.0's typical implementation has the following limitations

compared to an implementation of this specification. Trusted Platform Module 2.0 has limited support for the individual removal of the keys from the Trusted Platform Module. The application can only remove keys from the Trusted Platform Module by writing to the individual bits of a bitmap type of non-volatile storage, and use the status of a bit is not set to one as the requirement for the usage of a key. This limits the ability of the Trusted Platform Module to manage a significant amount of keys since the Trusted Platform Module has finite non-volatile persistent storages. Additionally, the Trusted Platform Module lacks the ability to safely migrate the keys stored on it to another Trusted Platform Module without knowing the Trusted Platform Module in advance, thus rendering it less useful for long term key storage. Finally, the type of algorithms supported is limited by the specific device, and applications using unsupported algorithms cannot protect their sensitive key material with a Trusted Platform Module.

Key Status Attest

Key Status Attest is a procedure to certify that a specific key is managed by a trusted computing device. This allows a remote party to recognize a cryptology key is suitable for a certain purpose without the need for physical inspection. There are a significant number of trusted computing solutions that support this use case including Trusted Platform Module 2.0.

This function is supported in the trusted computing platform with session mechanics. By creating a session with the vendor, the vendor can receive the peer certificate that attests to the identity of the trusted computing platform communicating with it. After this proof of identity is provided, the vendor will then transmit sensitive data to the trusted computing platform on the device, which can be a recognized private key.

Enclave

An enclave is an execution environment that allows software to be executed without interference or tampering. An enclave allows vendors to deploy sensitive logic to the user device without losing control or confidence in the result. By supporting this functionality, the software vendor could create games with better anti-cheat without an invasive kernel-mode driver while delivering a fair and balanced experience for the players, banks would be able to create a digital currency wallet that processes transactions offline, and instantly. This functionality is not supported by any discussed multi-tenant trusted computing solutions. Intel SGX(Costan and Devadas (2016)) supports this functionality, however, it is not accessible to uncoordinated vendors.

This function is supported by the trusted computing platform with web assembly

programmable logic. This programmable logic follows a set of verification rules that verify the integrity of the web assembly executable binary before executing it in an isolated environment inaccessible to other parts of the system.

Protected Input and Output

Protected Input and Output is a method of collecting input or displaying content without interference from the rest of the system. This prevents a compromised ordinary execution environment from intercepting the communication between the trusted computing platform and the user. Without this protection, even if the encryption key is protected, the data presented to the user still could be captured by the malicious application running in the ordinary execution environment. With this functionality, use cases such as financial transaction confirmation, or end to end protected communication can be realized. This functionality is not present on any discussed trusted computing solution.

The trusted computing platform supports this with Protected Input and Output. The trusted computing platform can collect input from input devices by taking control directly, or vice visa for output. The protected input and outputs are never available to the client in plaintext, the trusted computing client need to rely on the trusted computing platform to interact with these encrypted containers. This makes it significantly harder for a compromised ordinary execution system to obtain the sensitive user data exchanged with the user directly.

5.2 Attack Analysis

A trusted computing platform based on this specification does not have design vulnerabilities. In these sections, different kinds of attacks to the system are analysed, and their respective countermeasures are described.

5.2.1 Vendor Information Extraction

Firmware may be used to process sensitive information from its vendor. An attacker may attempt to extract data from it in order to gain access to information otherwise unavailable to trusted computing clients. The sensitive information is typically stored in the non-volatile storage of a trusted firmware, and transmitted with a session associated with the vendor. In both cases, the attacker needs to obtain the inputs or outputs that are ordinarily inaccessible to it. However, the firmware vendor can use mandatory constraints in the manifest to prevent this kind of attack.

Trusted Platform Module 2.0 supports the protection of cryptography keys, however, the sensitive information protected with these keys have to be processed in an ordinary execution environment since Trusted Platform Module cannot execute logic supplied by the client. Smart cards and ARM Trusted Execution Environment can complete the same function, but only the device manufacturer or its approved vendor can use this functionality.

Output Type Modification

The attacker can attempt to modify the type of output from a session or non-volatile storage to plaintext in order to obtain information ordinarily unavailable to the trusted computing client. This can be prevented with a type constraint in the manifest. If such a constraint is not found in the restriction list submitted by the client, then the invocation will fail due to mandatory constraints violations. If such a constraint is submitted by the client, then the invocation will fail due to the constraint not being met.

Output Session Destination Modification

The attacker can attempt to modify the session's destination by modifying the destination of the session to redirect the sensitive output to another trusted computing device controlled by the attacker. This can be prevented by creating mandatory session constraints with remote certificate hash anchored to the software vendor. The attacker cannot modify details of constraints since every constraint is compared to the manifest, and is required to be bit-identical to be considered a match. The session output emits a value of remote certificate hash as a property of the output. This value will be used to enforce this constraint.

Input Type Modification

The attacker can attempt to use non-volatile storage with sensitive data as an input if the firmware will echo that input. Typically, this kind of input is used as a way for the client to submit its own data to the client. The firmware vendor can prevent the attack by creating mandatory type constraints of the input as plaintext. The attacker will not be able to use this input as a way to get a copy of sensitive data stored within the non-volatile storage. Similarly, the attacker can attempt to change a random number input into a plaintext input, and supply it with known information to manipulate the internal processing of the firmware. Cryptography primitives such as Elliptic Curve Digital Signature Algorithm require high-quality random numbers, and a non-unique random number can result in the leak of private keys.

5.2.2 User Information Extraction

The trusted computing platform is designed with the protection of user information in mind. Before the user provides any sensitive information to the application via the trusted computing platform, the user will be able to review the usage constraint of the application. A malicious application may attempt to use a user data in a way other than the original usage the user has approved it for. The trusted computing platform uses input constraints for user input and a set of verification rules to prevent the abuse of user data.

Existing Trusted Computing solutions like Trusted Platform Module 2.0 or Smart Card does not support the protection of user data since it cannot typically interact with users directly, instead, all data need to pass through a potentially compromised ordinary execution environment. An attacker with full access to an ordinary execution environment can intercept any information received from or send to users. This prevents these Trusted Computing solutions from effectively protecting the end user's data from a compromised ordinary execution environment.

Mismatch of Input Data and Input Restriction Hint

During the invocation, if a user input is used as a input, the client will need to submit the usage restrictions associated with it. An attacker may attempt to submit a different set of restrictions from the one declared when collecting users' input. However, the trusted computing platform specification is designed to prevent this kind of attack by storing a digest of restrictions alongside the data. If the input restriction hint does not match the digest stored within the data packet itself, the invocation will fail.

Mismatch of Input Restriction and Input Restriction Hint

All restrictions appearing in the Input Restriction Hint are required to be present in the Input Restriction. This makes sure that the restriction required by the input is applied to the input, and validated as a part of the invocation process. An attacker could attempt to remove some of the constraints or change the constraints, however, the trusted computing platform requires the constraints to be present and bit-identical to the corresponding part of Input Restriction Hint. Otherwise, the invocation will fail.

Mismatch of Input Restriction Content

The restrictions associated with input can contain restrictions like Output At Least that limits the restrictions of all outputs to at least contain certain restrictions. These restrictions make sure that the usage of data satisfies certain requirements. Other restrictions

may also exist. The attacker may attempt to use the input in an invocation that does not confirm these restrictions, however, the invocation will fail.

5.2.3 Replay Attack

The trusted computing platform is designed to prevent replay attacks. By utilizing the non-volatile storage facility within the trusted computing platform, developers can create non-reversible interactions that have persistent effects. For a trusted computing platform, the removal of instances and the read and write of non-volatile are non-replayable interactions.

Trusted Platform Module 2.0 has limited protection against replay. Only password authentication can have a side effect of increasing pin fail non-volatile counters. All other operations have no protection from the replay attack as an attacker can otherwise retry indefinitely.

Reinstalling a Firmware to Restore Status

Uninstalling a trusted computing firmware is permanent and irreversible. This action can be utilized by the user to quickly destroy all data associated with a trusted computing firmware. This action is useful when the user is transferring the device to another individual or is under imminent threat. Each instance is unique since it contains a randomly generated secret stored within the Trusted Computing Platform. This secret cannot be recovered once removed during instance uninstallation. Every resource associated with the instance is encrypted and authenticated with this secret. The attacker can try reinstalling the application to recover the data. However, no data associated with the previous instance will be accessible since a different key will be used to process associated data.

Retry Invocation by Replay Non-Volatile Storage

The Trusted Computing Platform does not store the content of non-volatile storage within its protected storage. These data are instead provided by the invocation client's operating system. The instance stores a digest of an encrypted version of the content of non-volatile storage. The encrypted version of the content of non-volatile storage is then managed by the operating system. The attacker may attempt to restore this data to a previous state in order to double-spend money or otherwise reverse an operation. However, since the trusted platform will verify the digest of data against the record stored within its own protected storage. If the data stored in the external storage does not match the record, it will not be loaded.

Deprive Power During Execution

The Trusted Computing Platform requires electricity to function. However, it does not have the ability to control the electricity supply. If the electricity is deprived during execution, then it may lose some of the most recent states. The attacker may attempt to deprive electricity to the Trusted Computing Platform during the execution to prevent the new state from being written to the persistent storage. However, the Trusted Computing Platform prevents this by always writing to the persistent storage before providing feedback for invocations. The attacker has no way to gain any additional information before the digest of new non-volatile storage is written to the protected storage. This makes depriving the power during the execution ineffective for achieving the intended goal of reversing interactions, since the result of the interactions is not available until the change is persisted.

5.2.4 Tampering

The Trusted Computing Platform has a set of asset verification steps that ensures the Trusted Computing Platform is executing the firmware created by the software vendor, and loading states generated by the Trusted Computing Platform itself. This is accomplished by a set of verification rules that verify the contents supplied from external components. This is required since the Trusted Computing Platform has limited storage capacity, and requires to load firmware definitions from an external source to function.

Modifying Firmware Manifest

The firmware manifest contains important data about firmware. It contains the restrictions applied to the firmware and the digest of executable logic. The attacker could attempt to modify the manifest. However, the digest of the manifest file is stored within a certificate signed by the vendor. The attacker has no access to the private key required to issue the certificate. The attacker, thus, cannot create a signed manifest accepted by the Trusted Computing Platform.

Repack Firmware Manifest

The firmware manifest is self-signed. This allows any vendor to create firmware, including attackers. From the perspective of the Trusted Computing Platform, it cannot tell the difference between firmware from the valid vendor, or firmware created by an attacker. However, the session peer for any given session is required to hold a certificate from the vendor or an accredited Trusted Computing Platform manufacturer. The attacker could

attempt to create a repacked version of the firmware manifest signed with the attacker's key. However, this firmware cannot receive data from the original vendor since the vendor certificate does not match. This can prevent the strategy of repacking the firmware manifest from being useful for receiving sensitive information from the firmware vendor.

Modifying Firmware Executable Code

For each firmware, there is an executable binary associated with it. Since it contains all the executable logic for the firmware, its content's integrity is critical to the correct operation of the firmware. The digest of the executable code is stored within the signed manifest. The attacker could attempt to modify the content of the firmware. However, the modified version of it will be rejected by the Trusted Computing Platform during the execution, as the Trusted Computing Platform verifies the firmware against the digest before execution.

5.3 Limitations

5.3.1 Restriction Granularity

In the current version of the specification, there is an insufficient amount of type of restrictions. This makes it possible that some marginal use cases will be infeasible with the current set of restrictions. Adding more types of restrictions will make it easier to make precise and useful restrictions on the input and output.

5.3.2 Application Binary Interface Efficiency

In the current version of the specification, the application binary interface of the executable logic is not optimized. The current Concise Binary Object Representation based encoding method requires a significant number of client libraries to support the parsing of these structures, which consumes valuable memory space within the Trusted Computing Platform. Additionally, In the proof of concept implement, the communication between Trusted Computing Platform and executable firmware is conducted with The WebAssembly System Interface. This interface has so many features that are not strictly needed by the Trusted Computing Platform and requires extensive programming to supports all operations required. Both of them make it significantly harder to implement the Trusted Computing Platform efficiently.

5.3.3 Lack of Physical Implementation

The proof of concept implementation of this specification is based on software emulation. This kind of emulation does not provide full information about the usage characteristics of the application. Despite, in theory, this Trusted Computing Platform specification is theoretically possible to be implemented on a physical device to provide a secure and general purpose trusted computing device. However, due to time constraints, this is not realised. A physical implementation can provide better information on how this specification work under the real-world scenario.

Physical devices like Android device already includes ARM Trusted Execution Environment. It has all the basic building blocks for creating a Trusted Computing Platform, including an isolated execution environment, an replay protected memory block, an internal random number generator, and the ability to control input and output devices directly. Currently, these protections are used to make application-specific functionalities, such as password retry limit(Google and et al (2020a)) , android protected confirmation(Google and et al (2020d)) , or Digital Rights Management restriction enforcement. Once a Trusted Computing Platform is implemented on such a platform, these existing trusted computing facilities will be accessible to all applications.

Chapter 6

Discussion

6.1 Implementation Considerations

6.1.1 Device Manufacturer Recognition

Each Trusted Computing Platform maintains a list of recognized device manufacturers for the purpose of processing protected user data. This is required since only recognized Trusted Computing Platforms should be trusted with enforcing the restrictions. However, the update strategy of this list may need to be carefully considered. This includes whether to make entities on this list revocable. A revocable accreditation can respond to the discovered implementation flaw of the Trusted Computing Platform swiftly, however requires frequent network communication with the manufacturer's server. The precise criteria for the inclusion and exclusion of recognized Trusted Computing Platforms also need to be considered to make sure no inconvenience is caused to the user while protecting the integrity of the Trusted Computing Platform ecosystem.

6.1.2 Ordinary Execution Environment Support

It is likely that some of the functionality provided by a Trusted Computing Platform requires support from the operating system running within the ordinary execution environment. This includes the storage of non-violate storage, session tickets, and the manifest data for installed applications. This data is required to be managed by the support software running in the ordinary execution environment. With the support from the ordinary execution environment operating system, the clients' interaction with a trusted computing environment will require less state management, and it would be less likely for the firmware to enter an unrecoverable state as a result of failure to maintain the status.

6.2 Social Impact

The Trusted Computing Platform can create social impacts from its software design. Since the specification makes it significantly easier to protect the data protection rights of users, applications will need to justify the reason to collect user data that is not protected by a Trusted Computing Platform.

This kind of change removes the ability of an application to collect sensitive user data for profit. Thus, an alternative monetization scheme may be required for some applications. This can assist in the transition away from the user information based monetization model.

It is expected that the implementation of this specification will receive pressure from intelligence agencies and law enforcement since it would reduce their ability to perform mass surveillance and search. Since device manufactures are required to receive recognition for their device to receive protected session data from other device manufacturers, a system like Web Public Key Infrastructure can be created to verify the compliance of specification and the protection of user data. If device manufacture is misbehaving, it will lose recognition from other device manufactures and their device will no longer functioning correctly. This can be used to deter device manufacture from misbehaving. In the worst case, this specification does not give device manufacture or their affiliated attacker any additional access to the users' information, since they already have full access to user information on their device.

6.3 Adoption

The adoption of this Trusted Computing Platform should start with an embedded environment or other single-purpose devices with a limited amount of vendors. This could allow this specification to be used in practical settings while retaining the ability to update the specification and tolerate the lack of certain functionalities.

The general adoption of this specification will take place in smartphones or other integrated electronic devices. Smartphones come with touchscreens and buttons that are directly integrated with the system. This will allow the device manufacturer to create a full functionality implementation of this specification without cooperation from peripheral manufacturers. For desktop devices, the output device like screens or input device like mouse and keyboard does not provide an encrypted communication channel and requires drivers running in ordinary execution environment for interaction.

The final phase of the adoption will take the form of application vendors creating and using this Trusted Computing Platform as a way to provide more functionality to

its users. This will encourage users to expect the application to provide fully transparent and auditable end to end security for the user data.

Chapter 7

Conclusions & Future Work

In this paper, a systemic method of creating a Trusted Computing Platform is defined. It has the properties that it can be shared between multiple uncoordinated software vendors, that can support all kinds of logic's in a general manner with Turing complete execution engine, that promotes the user's data protection right by enforcing the transparency on the transfer and usage of user data.

This specification defined a Trusted Computing Platform that executes executable logic in an isolated environment that have protected memory, non-violate memory, and execution context. It accepts the manifest that declares how should the Trusted Computing Platform interact with the programmable logic of the trusted computing firmware, and the resources required by the firmware. The client running on a separate execution environment will send a request to the trusted execution environment, where the execution happens after validating the invocation request based on the manifest associated with the trusted computing firmware. The input and outputs of the invocation are managed by a trusted execution environment according to the firmware.

A proof of concept implementation was created to demonstrate the functionality of the Trusted Computing Platform specification. This application is created with the Go programming language and other third-party libraries. This implementation is a software emulator of the protocol. A demo application is created to showcase the functionality provided by the Trusted Computing Platforms. This application assisted the creation of a full end to end communication system, with a fully transparent and auditable user data usage restriction. The user can observe the usage restriction of data before inputting any data.

7.1 Future Work

7.1.1 Physical Device Implementation

Currently, there is no physical device implementation of this specification. This means currently all Trusted Computing Platform implementation only provides an emulation of the API, but do not provide security proprieties of the Trusted Computing Platform. Implementing on a physical device pose an additional challenge since this would require additional considerations about the resource usage and programming environments. Creating a physical device implementation can enhance the understanding of this Trusted Computing Platform proposal, and begin to build an ecosystem around this specification.

7.1.2 Additional Restriction Types

Currently, the type of restrictions is relatively limited. In the future version of the project need to have more type of restrictions in order to allow the Trusted Computing Platform to make precise and flexible restrictions on the usage of data, and archive more functionalities.

7.1.3 Application Binary Interface Efficiency

Currently, the proof of concept implementation of the application does not have its own optimised application binary interface for this particular purpose. The current Web Assembly System Interface and Concise Binary Object Representation based input and output format require a significant amount of support libraries as they require unused functionalities. In the future, a more purpose-built application binary interface would make it significantly easier to implement this specification in the hardware, and reduce memory requirement for the trusted computing platforms.

7.1.4 Production Rollout

Eventually, A full-featured smartphone with support for this specification can be created. In this case, all functionalities designed in the specification will be presented on this implementation.

An ecosystem of applications that utilize this trusted computing specification can also be created and promoted. These applications will utilize the trusted computing facilities provided by this specification. This can evaluate the effectiveness of this specification in production environments.

Bibliography

Alimi, V. and Pasquet, M. (2009). Post-distribution provisioning and personalization of a payment application on a uicc-based secure element. In *2009 International Conference on Availability, Reliability and Security*, pages 701–705. IEEE.

Android Open Source Project and et al (2020a). Application Signing. Retrieved August 15, 2021, from `https://source.android.com/security/apksigning`.

Android Open Source Project and et al (2020b). Key and ID Attestation. Retrieved August 15, 2021, from `https://source.android.com/security/keystore/attestation`.

Arthur, W., Challener, D., and Goldman, K. (2015). *A practical guide to TPM 2.0: Using the new trusted platform module in the new age of security*. Springer Nature.

Bellare, M., Rogaway, P., and Wagner, D. A. (2003). Eax: A conventional authenticated-encryption mode. *IACR Cryptol. ePrint Arch.*, 2003:69.

Bhargavan, K., Blanchet, B., and Kobeissi, N. (2017). Verified models and reference implementations for the tls 1.3 standard candidate. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 483–502. IEEE.

Bicakci, K., Ak, I. K., Ozdemir, B. A., and Gozutok, M. (2019). Open-tee is no longer virtual: Towards software-only trusted execution environments using white-box cryptography. In *2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pages 177–183. IEEE Computer Society.

C. Bormann, Universität Bremen TZI, P. Hoffman, and ICANN (2020). Concise Binary Object Representation (CBOR). Retrieved August 15, 2021, from `https://datatracker.ietf.org/doc/html/rfc8949`.

Chen, Z. (2000). *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*. Addison-Wesley Longman Publishing Co., Inc., USA.

Chow, S., Eisen, P., Johnson, H., and Van Oorschot, P. C. (2002). White-box cryptography and an aes implementation. In *International Workshop on Selected Areas in Cryptography*, pages 250–270. Springer.

Costan, V. and Devadas, S. (2016). Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016(86):1–118.

Costan, V., Sarmenta, L. F. G., van Dijk, M., and Devadas, S. (2008). The trusted execution module: Commodity general-purpose trusted computing. In Grimaud, G. and Standaert, F.-X., editors, *Smart Card Research and Advanced Applications*, pages 133–148, Berlin, Heidelberg. Springer Berlin Heidelberg.

D. Cooper and et al (2008). Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Retrieved August 15, 2021, from `https://datatracker.ietf.org/doc/html/rfc5280`.

Google and et al (2020a). Gatekeeper. Retrieved August 15, 2021, from `https://source.android.com/security/authentication/gatekeeper`.

Google and et al (2020b). Introduction to gRPC - An introduction to gRPC and protocol buffers. Retrieved August 15, 2021, from `https://grpc.io/docs/what-is-grpc/introduction/`.

Google and et al (2020c). Language Guide (proto3). Retrieved August 15, 2021, from `https://developers.google.com/protocol-buffers/docs/proto3#json`.

Google and et al (2020d). Protected Confirmation. Retrieved August 15, 2021, from `https://source.android.com/security/protected-confirmation`.

Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., Wagner, L., Zakai, A., and Bastien, J. (2017). Bringing the web up to speed with webassembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, page 185–200, New York, NY, USA. Association for Computing Machinery.

Hua, Z., Gu, J., Xia, Y., Chen, H., Zang, B., and Guan, H. (2017). vtz: Virtualizing arm trustzone. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 541–556.

International Organization for Standardization (2007). Identification cards — Integrated circuit cards — Part 2: Cards with contacts — Dimensions and location of the contacts. Standard, International Organization for Standardization, Geneva, CH.

International Organization for Standardization (2011). Identification cards — Integrated circuit cards — Part 1: Cards with contacts — Physical characteristics. Standard, International Organization for Standardization, Geneva, CH.

International Organization for Standardization (2018). Cards and security devices for personal identification — Contactless proximity objects — Part 1: Physical characteristics. Standard, International Organization for Standardization, Geneva, CH.

Jacomme, C., Kremer, S., and Scerri, G. (2017). Symbolic models for isolated execution environments. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 530–545.

James Bottomley (2017). tpm: expose spaces via a device link /dev/tpmrm<n>. Retrieved August 15, 2021, from `https://github.com/torvalds/linux/commit/fdc915f7f71939ad5a3dda3389b8d2d7a7c5ee66`.

Jarkko Sakkinen (2017). in-kernel resource manager. Retrieved August 15, 2021, from `https://lwn.net/Articles/716259/`.

Johnson, D., Menezes, A., and Vanstone, S. (2001). The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63.

Kaur, G. and Fuad, M. M. (2010). An evaluation of protocol buffer. In *Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon)*, pages 459–462.

Lehtonen, S. J. et al. (2020). Comparative study of anti-cheat methods in video games. (Master's thesis, University of Helsinki, Helsinki, Finland). Retrieved from `http://hdl.handle.net/10138/313587`.

Li, W., Xia, Y., and Chen, H. (2019). Research on arm trustzone. *GetMobile: Mobile Comp. and Comm.*, 22(3):17–22.

Mitchell, C. (2005). *Trusted Computing (Computing and Networks)*, volume 6. The Institution of Engineering and Technology.

Mukhtar, M. A., Bhatti, M. K., and Gogniat, G. (2019). Architectures for security: A comparative analysis of hardware security features in intel sgx and arm trustzone. In *2019 2nd International Conference on Communication, Computing and Digital systems (C-CODE)*, pages 299–304. IEEE.

Nurseitov, N., Paulson, M., Reynolds, R., and Izurieta, C. (2009). Comparison of json and xml data interchange formats: a case study. *Caine*, 9:157–162.

Open Mobile Terminal Platform (2009). Advanced Trusted Environment:OMTP TR1. Retrieved August 15, 2021, from https://www.gsma.com/newsroom/wp-content/uploads/2012/03/omtpadvancedtrustedenvironmentomtptr1v11.pdf.

Park, S., Ahmad, A., and Lee, B. (2020). Blackmirror: Preventing wallhacks in 3d online fps games. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 987–1000, New York, NY, USA. Association for Computing Machinery.

Reddy, A. K., Paramasivam, P., and Vemula, P. B. (2015). Mobile secure data protection using emmc rpmb partition. In *2015 International Conference on Computing and Network Communications (CoCoNet)*, pages 946–950.

Smith, S. W. (2013). *Trusted computing platforms: design and applications*. Springer.

Sun Microsystems, I. (1998). *Java Card Applet Developer's Guide*. Sun Microsystems, Inc.

Syrus Akbary and et al (2020). Wasmer. Retrieved August 15, 2021, from https://github.com/wasmerio/wasmer.

Trusted Computing Group (2013). Trusted Platform Module Library - Part 1: Architecture. Retrieved August 15, 2021, from https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-1-Architecture-00.96-130315.pdf.

Trusted Computing Group (2016). Trusted Platform Module Library - Part 3: Commands. Retrieved August 15, 2021, from https://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-3-Commands-01.38.pdf.

Wang, J., Xie, X., Wang, Q., Yan, F., Hu, H., Zhou, S., and Wang, T. (2013). Towards a trusted launch mechanism for virtual machines in cloud computing. In *International Conference on Cloud Computing*, pages 90–101. Springer.

Zhang, L., Chen, L., Zhang, H., and Yan, F. (2007). Trusted code remote execution through trusted computing and virtualization. In *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007)*, volume 1, pages 39–44. IEEE.

Zhao, S., Zhang, Q., Qin, Y., Feng, W., and Feng, D. (2019). Sectee: A software-based approach to secure enclave architecture using tee. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 1723–1740, New York, NY, USA. Association for Computing Machinery.