View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by The University of North Carolina at Greensboro

SAMA, VIVEK REDDY, M.S. Universally Composable Zero Knowledge Protocol using Trusted Platform Modules. (2011)
Directed by Stephen Tate. 36 pp.

Cryptographic protocols that are established as secure in the Universally Composable (UC) model of security provide strong security assurances even when run in complex environments. Unfortunately, in order to achieve such strong security properties, UC protocols are often impractical, and most non-trivial two-party protocols cannot be secure in the UC model without some sort of external capability (or "setup assumption") being introduced. Recent work by Hofheinz *et al.* [18] provided an important breakthrough in designing realistic universally composable two party protocols, in which they use trusted, tamper proof hardware as a special type of helping functionality which they call a *catalyst*. Hofheinz *et al.* use government issued signature cards as a catalyst to design universally composable protocols for zero-knowledge proofs and commitments, but did not give a complete security proof for either protocol.

In this thesis, we consider another form of security hardware, Trusted Platform Modules (TPMs), which are more widespread than signature cards and are currently shipped as a part of almost every business laptop or desktop. Trusted Module Platforms are tamper evident devices which support cryptographic functionalities including digital signatures, but have a different key management model from signature cards. In this thesis we consider TPMs as *catalysts* and describe a universally composable zero knowledge protocol using Trusted Platform Modules. We also present a complete security proof for both the Hofheinz's universally composable zero knowledge protocol from signature cards and our universally composable zero knowledge protocol using TPMs as a *catalyst*.

# UNIVERSALLY COMPOSABLE ZERO KNOWLEDGE PROTOCOL USING TRUSTED PLATFORM MODULES

by

Vivek Reddy Sama

A Thesis Submitted to
the Faculty of the Graduate School at
The University of North Carolina at Greensboro
in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Greensboro
2011

Approved by

_____
Committee Chair

© 2011 by Vivek Reddy Sama

*To my parents and friends*

APPROVAL PAGE

This thesis has been approved by the following committee of the Faculty of The Graduate School at The University of North Carolina at Greensboro.

Committee Chair _____
Stephen Tate

Committee Members _____
Shanmugathasan Suthaharan

_____
Lixin Fu

_____
Date of Acceptance by Committee

_____
Date of Final Oral Examination

ACKNOWLEDGEMENTS

It is a pleasure to thank all those who made this thesis possible.

First, I would like to express my sincere gratitude to my adviser, Dr. Stephen Tate, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding on my thesis. His guidance helped me all the time of research and writing of this thesis. I would like to thank my committee members Dr. Shan Suthaharan and Dr. Lixin Fu for their support and advice, which helped me to refine my work further.

I also thank my family members for being there with me and supporting in all the ways without which, this was not possible. My special thanks to all my friends at Warren Street and University village.

TABLE OF CONTENTS

Page

CHAPTER

I. INTRODUCTION . . . . . 1

1.1. Zero-Knowledge Proofs . . . . . 2
1.2. Universal Composable Framework . . . . . 6
1.3. Trusted/Tamper Proof Hardware . . . . . 9
1.4. Prior Work . . . . . 13
1.5. Motivations and Results Overview . . . . . 15

II. UC ZERO-KNOWLEDGE PROTOCOLS . . . . . 17

2.1. Zero knowledge protocols based on Signature Cards . . . . . 17
2.2. Zero-knowledge protocols based on TPMs . . . . . 25

III. CONCLUSION AND FUTURE DIRECTIONS . . . . . 33

REFERENCES . . . . . 35

# CHAPTER I

# INTRODUCTION

The term "cryptography" is derived from the Greek phrase for "hidden writing" [22]. The story of cryptography goes back thousands of years, and initially cryptography dealt with the methods of encryption which used only pen and paper. The development of cryptanalysis (the science of breaking secret writing) has been balanced by the development of cryptography. The first glimpses of modern cryptography were seen during the phases of World War I and World War II, and over the years, the use of cryptography has drastically increased in military applications. Until 1970's, the domain of cryptography comprised of only encryption and decryption techniques, i.e., keeping messages secret in storage or transmission. Working with data in complex environments, the protection of data may have the traditional meaning of secrecy (confidentiality) but may also include goals such as integrity, authenticity or fairness, and so cryptography has expanded to cover techniques that support goals other than secrecy. For example, the authenticity of data in a complex environment can be addressed using digital signatures [21]. In the 1970's and 1980's breakthroughs in computer science fundamentally changed the way cryptography was viewed. In the 1970's, the data encryption standard (DES) [24] and public-key cryptography [10] were introduced. The invention of the key exchange mechanism by Diffie and Hellman [10] in 1976, and the first public key encryption algorithm by Rivest, Shamir and Adleman [21] in 1978 played a vital role in building the new idea of public key cryptosystems. In 1984, the concept of an interactive proof system was introduced

by Goldwasser *et al.* [14]. The proposed systems led to the development of new computational models of security. Later, many additional cryptographic techniques and concepts were introduced, including one-way functions, pseudo-random generators, zero-knowledge proofs [13] and witness indistinguishable arguments [11]. Unlike the classical view of cryptography, which focused on only secret messages, modern cryptography has a much expanded domain.

In this work we present a zero-knowledge protocol using Trusted Platform Modules (TPMs) to achieve universal composable security. This chapter mainly focuses on background information, which sets a foundation for us to present our work. In Section 1.1, we present background information on zero-knowledge proofs. The universal composable model of security is discussed in Section 1.2. We discuss protocols related to trusted/tamper proof hardware in Section 1.3. Prior work and an overview of our results are given in Section 1.4 and Section 1.5 respectively.

## 1.1 Zero-Knowledge Proofs

In 1985, Goldwasser *et al.*[13] introduced a new concept called zero-knowledge proof, which is an interactive protocol between two parties, prover $P$ and verifier $V$, where $P$'s job is to convince $V$ that a common input $x$ is in some language $L$. For example, $x$ might be a graph, and $P$ needs to convince $V$ that $x$ contains a Hamiltonian cycle. In a zero knowledge proof $P$ proves to $V$ that a statement is *true*, without revealing any information other than the veracity of the statement. Formally, $P$ and $V$ are probabilistic Turing machines such that at the end of this interactive protocol, $V$ outputs either "Accept" or "Reject" as a result. To be an interactive proof, this protocol should have two properties: It should be *complete*,

where an honest prover will always gets acceptance, and it should be *sound*, meaning that no cheating prover can convince an honest verifier that a false statement is *true*, except with small probability.

**Definition I.1. Interactive Proof System**

A pair of interactive machines $\langle P, V \rangle$ with prover $P$ and verifier $V$ is called an interactive proof system for a language $L$ if machine $V$ is polynomial-time and the following two conditions hold:

Completeness: For every $x \in L$, $Pr\left[\langle P, V \rangle(x) = 1\right] \geq \frac{2}{3}$

Soundness: For every $x \notin L$ and every interactive machine $B$, $Pr\left[\langle B, V \rangle(x) = 1\right] \leq \frac{1}{3}$

To be a zero-knowledge proof, it also should have a property called zero knowledge, where no cheating verifier learns any information other than statement to be proven. To illustrate the zero knowledge proof concept, we consider an example with participants Alice and Bob. Consider a situation in which Alice knows a Hamiltonian cycle $C$ for a large graph $G$. She wants to prove to Bob that $G$ has a Hamiltonian cycle, but does not want to reveal the cycle to him. Alice posts $G$ publicly such that both Alice and Bob share a common input graph $G$. Alice creates a graph $H$ isomorphic to $G$ by randomly permuting the vertices of $G$. She commits $H$ but doesn't reveal H to Bob. Lets say she writes each edge of $H$ on a slip of paper and turns them over so that they are visible to both of them so that Alice cannot change anything about $H$ at a later part of the protocol. Bob flips a coin. If heads, Alice shows how $H$ is isomorphic to $G$ by turning all pieces of paper that she put on the table. If tails, Alice shows the Hamiltonian cycle in $H$ to Bob by just turning over just the edges in the Hamiltonian cycle. This interactive game of Alice and Bob works accordingly with

the above mentioned thee properties, i.e., completeness, soundness and zero knowledge. In the above scenario, the Hamiltonian cycle $C$ is secret information, sometimes called a witness to the fact that $G$ has a Hamiltonian cycle, and verifier $V$ does not gain any information about the witness even at the end of the protocol.

In order to define the security of an interactive proof protocol we have to consider the following setting. Consider a cheating verifier $V^*$ interacting with $P$ on input $x \in L$. Since $P$ and $V^*$ are probabilistic, the output of $V^*$ is a random variable for each $x \in L$, or is an ensemble, which we denote $R_{x \in L}$ (this is "real" output). Next, consider creating a simulator $M_{V^*}$ which simulates the cheating verifier $V^*$ with same input $x \in L$ and generates the ensemble $S_{x \in L}$. Since $M_{V^*}$ does not interact with $P$, it can't discover $P$'s information, so if $M_{V^*}$ behaves essentially the same as $V^*$ interacting with $P$, then $V^*$ similarly can not gain information. Technically, this holds if $R_x$ and $S_x$ are computationally indistinguishable which means that for every probabilistic polynomial-time algorithm $D$, for every polynomial $p(.)$, and for all sufficiently large $|x|$, it holds that

$|Pr[D(x, R_x) = 1] - Pr[D(x, S_x) = 1]| < \frac{1}{p(|x|)}$.

**Definition I.2. Zero-knowledge Proof**

Let $\langle P, V \rangle$ be an interactive proof system for a language $L$. We say the proof system $\langle P, V \rangle$ is zero knowledge if for every expected polynomial time interactive Turing machine $V^*$ , there exists an ordinary expected polynomial time machine $M_{V^*}$ such that the probability ensembles $\{M_{V^*}(x)\}_{x \in L}$ and $\{P(x), V^*(x)\}_{x \in L}$ are polynomial indistinguishable.

The zero-knowledge property is not necessarily preserved in a scenario where mul-

tiple zero-knowledge protocol instances are executed simultaneously (in parallel) in the same environment [12]. To preserve the privacy in such a scenario, we consider a notion called Witness Indistinguishable proofs. This notion was introduced in 1990 by Shamir *et al.* [11] as a weaker variant of zero-knowledge proofs, but is still strong enough to be very helpful in providing security in some specific applications. In many interactive proof scenarios, there may be witnesses that prove that $x \in L$. In our previous Hamiltonian cycle example, any Hamiltonian cycle serves as a witness. The main idea of witness indistinguishable proof is to guarantee that the verifier will not be able to distinguish between provers that use different witnesses. In order to understand witness indistinguishablity, consider an example where Alice and Bob will be participating in an interactive proof system. Alice knows more than one Hamiltonian cycle to a graph $G$, say two Hamiltonian cycles $C$ and $D$. Both Alice and Bob share graph $G$ as common input. The goal for Alice is to convince Bob that it has a Hamiltonian cycle for graph $G$. Alice interacts with Bob using one of her Hamiltonian cycles, $C$ or $D$. Bob will verify the validity of the proof but his view on the $P$'s proof will be indistinguishable no matter which Hamiltonian cycle Alice uses. Therefore, the interactive proof between Alice and Bob is witness indistinguishable because Bob cannot tell which Hamiltonian cycle Alice uses to construct the proof.

Witness indistinguishabilty can be formally defined as follows.

**Definition I.3. Witness Indistinguishable Arguments of Knowledge (WIAOK)** Let $\langle P, V \rangle$ be an interactive proof system for a language $L \in \mathcal{NP}$ and $V^*$ be a probabilistic polynomial-time interactive machine and $R_L$ be a witness relation for a language $L$, i.e., if $x \in L$ there exist a witness $w$ such that $xR_Lw$. We say the proof system $\langle P, V \rangle$ is witness indistinguishable over $R_L$ if for any $V^*$ and every two

$w_x^1, w_x^2 \in R_L(x)$, the two ensembles $\{\langle P(w_x^1), V^* \rangle(x)\}_{x \in L}$ and $\{\langle P(w_x^2), V^* \rangle(x)\}_{x \in L}$ are computationally indistinguishable. In other words, for every probabilistic polynomial-time algorithm $D$, every polynomial p(.) and all sufficiently long $x \in L$, it holds that $Pr[D(x, \langle P(w_x^1), V^* \rangle(x) = 1] - Pr[D(x, \langle P(w_x^2), V^* \rangle(x) = 1]| < \frac{1}{p(|x|)}$.

In our work we assume that there exists witness indistinguishable arguments of knowledge (WIAOK) (which is true under the standard cryptographic assumption that one-way functions exist). WIAOK satisfy three properties. First, these arguments are computationally convincing proofs, i.e., a computationally limited prover can make the verifier accept high negligible probability. Second, they are arguments of knowledge, which means that from a prover which is giving convincing arguments a witness can be extracted. Third, the arguments of knowledge are witness indistinguishable: the protocol runs are indistinguishable for different witnesses.

### 1.2 Universal Composable Framework

Imagine a complex cryptographic environment with several types of protocols like identification protocols, key exchange protocols and several other interactive protocols. Lets say each type of protocol may be executed many times and each party in that environment may take part in several protocol executions simultaneously. Even if individual protocols are secure in isolation, one important question to ask is whether protocols retain security in such a complex environment. Lets say there exists a cheating party which participates in this environment and obtains valuable information by executing some protocols of that environment which it can use to compute responses of other protocols of the same environment. Traditional security definitions of cryptographic protocols treat protocols as stand-alone applications and

are helpful for simple analysis of protocols. These security definitions do not address security when these protocol instances are composed with other arbitrary protocol instances. Extending definitions of security of these protocols in such an environment is a way to address the issues with those cheating parties. Clearly, there is a need for a model of security which can better address security in such complex environments.

In 2001, Canneti proposed a framework called *Universal Composablity*, which guarantees that security defined for protocols as stand alone applications is preserved in the composition of these protocols instances with other arbitrary protocol instances in an unpredictable environment. A cryptographic functionality can be executed either in an "ideal world" in which all operations are performed by a completely trusted party (and hence the execution is secure), or in the "real world" where there is no trusted party and the parties have to execute the protocol just amongst themselves. The UC framework considers an algorithmic entity called the environment (observer) which gives inputs to the participating parties of a protocol (implementing some cryptographic task) and collects outputs from the parties of the protocol in course of its execution. Finally, the environment produces an output bit which can be interpreted as saying whether it thinks it has interacted with ideal functionality of a cryptographic task or a real protocol implementing same cryptographic task. Since the ideal world is inherently secure, if an environment cannot distinguish between this ideal functionality and a real protocol implementing the same cryptographic task, then that real protocol securely implements the ideal functionality, and we say that it is universally composable.

Consider the protocol $\pi$ that is executed in real world and ideal functionality $\mathcal{F}$ is works in ideal world. Formally, the participants of both worlds will be interactive

Turing machines, and we describe both worlds below.

**Ideal Model:** The participants in the ideal model are the environment $\mathcal{Z}$, ideal adversary $\mathcal{S}$, the ideal functionality $\mathcal{F}$, and dummy parties. $\mathcal{Z}$ can write inputs to these dummy parties and the outputs of these parties will be given to $\mathcal{Z}$. $\mathcal{Z}$ cannot see the communication between the $\mathcal{F}$ and dummy parties. Inputs given to dummy parties will be sent to $\mathcal{F}$ and anything received by $\mathcal{F}$ will be written to their local outputs. $\mathcal{S}$ communicates with $\mathcal{Z}$ and $\mathcal{F}$. $\mathcal{S}$ can corrupt parties such that it can see all the interactions between $\mathcal{F}$ and corrupted parties. There are no direct interactions between these dummy parties.

**Real Model:** In the real world, the protocol $\pi$ will be implemented by the parties interacting with each other. The participants of this world are environment $\mathcal{Z}$, real adversary $\mathcal{A}$ and participating parties. $\mathcal{Z}$ can write inputs to these real parties and the outputs of these parties will be given to $\mathcal{Z}$. Parties may communicate with each other in the course of the protocol, but $\mathcal{Z}$ cannot see these communications between the parties. $\mathcal{A}$ communicates with $\mathcal{Z}$ and $\mathcal{F}$. $\mathcal{A}$ can corrupt any parties in this protocol and $\mathcal{A}$ will participate in the protocol $\pi$ on behalf of any corrupted parties.

**Realizing an ideal functionality:** Protocol $\pi$ securely realizes ideal functionality $\mathcal{F}$ if for any real-life adversary $\mathcal{A}$ there exists an ideal adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and $\pi$ in the real model or with $\mathcal{S}$ and $\mathcal{F}$ in the ideal world.

Since universally composable security is a strict notion of security, designing realistic cryptographic tasks in universally composable framework need additionally helping functionalities. In earlier works, Hofheinz and Muller-Quade [17], and Backes

*et al.* [1] use helping functionalities random oracle and key registration authorities, respectively. These functionalities are defined in such a way that if a protocol instance implements some instance of helping functionality, then the helping functionality cannot be used by other applications without compromising UC security.

Hofheinz*et al.* [18] introduce the concept of catalyst, where a catalyst is a helping functionality which can be used simultaneously for different purposes without compromising universal composable security. $\mathcal{C}$ is a catalyst for functionality $\mathcal{F}$ if $\mathcal{C}$ can be implemented by $\mathcal{F}$ and the same instance $\mathcal{C}$ can be directly be used by other arbitrary applications without any additional precautions. The formal definition of a catalyst is given below.

**Definition I.4.** Let $\pi$ be a protocol realizing the functionalities $\mathcal{F}$ and $\mathcal{C}$ using $\mathcal{C}$. We say that $\mathcal{C}$ is used as a *catalyst* if $\pi$ realizes $\mathcal{C}$ by just relaying all requests and the respective answers directly to and from the functionality $\mathcal{C}$.

Hofheinz *et al.* use government issued signature cards as a catalyst in designing zero knowledge and commitments protocols in the universally composable framework. In our work, we use Trusted Platform Modules as a catalyst in designing a universally composable zero knowledge protocol.

### 1.3 Trusted/Tamper Proof Hardware

Unfortunately, fundamental limitations result in the impossibility of using the universally composable framework directly for most two-party protocols [7]. To counter this problem researchers have suggested adding some initial capability to the universal composable model (technically composed into the base model) which is called a "setup assumption." A variety of setup assumptions have been made to design two party

protocols in universally composed framework. Typical work in this area includes work by Canneti [3], Canneti and Fischlin [5], Lindell *et al.* [6], and Dodis *et al.* [4], who consider a setup where there exists one or more trusted parties that operate with some pre-agreed rules for all participants of the protocol. In practice, relying on such trusted parties is undesirable, making the design of realistic universally composable two party protocols very difficult. More recently, the design of two party protocols in the universally composable framework has been considered using assumptions of tamper-proof hardware, where each participant in the protocol will have access to its own such hardware. For example, Katz [19] proposed a physical setup assumption, specifically tamper-proof hardware tokens, which provides an interesting way to realize two-party protocols in the universally composable framework. With the advent of tamper evident hardware devices like TPMs and government issued signature cards, the design of protocols in the universally composable framework can consider realistic hardware assumptions. In this section, we present background information of Katz's tamper proof hardware tokens, signature cards and TPMs.

### Katz's Tamper Proof Hardware Tokens

As described above, Katz [19] was the first to consider tamper proof hardware tokens in designing universally composable multiple party protocols. The assumptions made by Katz in designing hardware tokens are as follows: First, each party participating in a protocol can construct its own tamper proof hardware token which implements a desired functionality. Second, when a hardware token is sent to some cheating party, it cannot deduce any information about the token's functionality except what it can observe from input/output behavior. Finally, when an honest party

is given a token from a cheating party, it cannot deduce any information about the functionality the token is implementing (it observes only outputs of the token). With these primitives, Katz [19] designed a protocol for universally composable multi-party commitments.

Subsequent work from Chandran *et al.* [8], Damgard *et al.* [9] improved on Katz's physical assumptions in various ways like including a resettable feature. Moragan and Segev [20] also improved Katz's solution by requiring only one of the participants of the two-party protocols to create a secure token.

**Signature Cards**

Hofheinz *et al.* [18] considered the use of government issues signature cards, which are tamper proof devices that function with globally accepted setup. Each card holds a legitimate key pair (public key and private key) where the private key is embedded into the card. The functionality of such cards includes generating digital signatures for messages sent to the card. The private key of a signature card cannot be extracted even by the legitimate owner of the card under any circumstances. Therefore, digital signatures generated by these cards are unforgeable by anyone without access to the card. Hofheinz *et al.* [18] use the cards as catalysts to design two party universally composable zero knowledge proof and commitment protocols. As some European governments are issuing such tamper evident signature cards, the design of realistic universally composable protocols is feasible. Drawbacks of the use such cards includes the fact that these signature cards are not widespread and the participants of the protocol must completely trust the producer/issuer of these cards (which is a threat).

**Trusted Platform Modules**

The Trusted Computing Group, a consortium of more than 100 companies, has defined a specification for a security chip called a Trusted Platform Module (TPM), which is designed to be attached to the motherboard of a computer. The functionality of a TPM includes the secure generation of cryptographic keys, generating pseudo-random numbers, remote attestation, sealed storage and common cryptographic operations like RSA encryption, RSA digital signatures and generating hashes (SHA1). A TPM is a tamper evident hardware device which is designed to software attacks.

Key management in TPMs is quite different from the signature cards described in the previous section. Users can create different RSA keys for different purposes. These user generated keys include bind keys, signing keys, legacy keys, storage keys and Attestation Identity Keys (AIKs). Creating and using a key in a TPM involves several steps. We give a rough idea of how it works below:

(1) Every user-created key is created under a parent key (which must be a storage key).

(2) When users request creation of a key, they must specify the parent key and a user selected secret called the authorization secret. The TPM returns a wrapped key blob which contains newly created key (both public key and encrypted private key, which is encrypted with the parent key). Since the private key never exists unencrypted outside a TPM, TPMs provide secure key management.

(3) In order to use a user generated key in the TPM, it has to be loaded into TPM so that the private key can be decrypted and used for purposes like signing (if

the loaded key is a signing key) or decryption (if the key is a bind key). For loading the key, the user has to provide the authorization secret for the parent key, which should be loaded prior to this operation. If the TPM successfully loads the key, it returns a key handle for that key which serves as a reference to the user for future purposes, and when it is used the user must provide the authorization secret to the TPM.

Each TPM is shipped with a unique Endorsement Key which is embedded in it by the TPM vendor. The validity of EK reflects the validity of TPM. Using the TPM's EK, a TPM owner can create Attestation Identity Keys (AIKs), which can be certified by trusted certificate authorities (CAs). AIKs can certify information coming from a TPM, including keys generated by the TPM. For detailed descriptions of functionalities of a TPM, we refer the reader to the TCG's specifications [15].

An important advantage of TPMs over other secure hardware proposals is that it is designed as an attachment to the motherboard of the local system, and TPMs are currently being shipped with most business-class laptops and desktops. This makes TPM a widespread device which can be helpful in designing realistic protocols in the universally composable framework.

## 1.4 Prior Work

In Section 1.3, we have discussed a number of setup assumptions, including trust assumptions and physical assumptions, for designing universally composable secure multiple party protocols. In this section we discuss some existing solutions and other related work.

Canneti and Fischlin [5] introduced the use of a Common Reference String (CRS)

model as a setup assumption in the universally composable framework. In this model, the CRS is generated by a trusted party which functions in a pre-agreed manner set by all parties participating in the protocol. For example, in some situations the requirements of CRS are that it should be a uniformly distributed random string. In practice, one must trust some party to honestly generate the CRS, which may not be realistic. Public key registration services, considered in by Barak *et al.* [2] and Dodis *et al* [4] also serve the purpose of designing multiparty protocols in universally composable framework with certain setup assumptions. For example, the functionality of Barak *et al.* [2] will not allow any cheating party to register its public key which is not generated with pre-agreed rules. Clearly the drawback in this model, is that it is difficult in practice to guarantee such a public key with the given setup assumption.

Each of the mentioned works use some trusted assumption that is unrealistic. Katz [19] introduces a new way to design protocols in the universally composable setting by suggesting a hardware assumption of tamper proof hardware tokens created by each party, which eliminates the necessity of trust in a third party. In 2005, Hofheinz *et al.* [18] introduced a feasible way of designing realistic zero-knowledge protocol and commitment protocols in the universal composable setting with assumptions of government issued signature cards. One drawback of this approach is that parties of the protocol must have complete trust in the producer or issuer of these signature cards (What if the producer/issuer is corrupted?). With the advent of TPMs, the scope of designing realistic multiple party protocols in the universal composable framework will be wider. Prior work using TPMs to design cryptographic protocols showed that TPMs can enable previously impossible functionality. For example, Gunupudi and Tate [16] created instantiation of a random oracle in multi-party settings where

each party has access to a TPM. Tate and Vishwanathan [23] used TPM capabilities and designed alogrithms to replace cut and choose protocols in verifiable encryption schemes, making the protocol non-interactive and offering more computational efficiency.

### 1.5 Motivations and Results Overview

The goal of this work is to consider universally composable zero knowledge using signature cards and TPMs. We give a complete security proof for Hofheinz *et al.*'s proposed universally composable zero knowledge protocols using signature cards, and we extend this protocol to propose a universally composable zero knowledge protocol using Trusted Platform Modules. Some motivations for this work are given below.

(1) Hofheinz *et al.* [18] did not give a complete security proof for their UC zero knowledge arguments using signature cards.

(2) The parties participating in Hofheinz *et al.* [18] must have complete trust in the producer or issuer of the card.

(3) The government issued signature cards are not widespread, i.e., not accessible to many users across the world.

The results of this work are as follows. First, we redefine the ideal functionality used in Hofheinz *et al.* [18] by using the traditional zero-knowledge ideal functionality. Secondly, we present a complete security proof for the universally composable zero knowledge protocol using signature cards [18]. Extending this work to a more common form of secure hardware, we define the TPM-based catalyst's ideal functionality, and

propose a universally composable zero knowledge protocol using TPMs. We also present a complete security proof for our protocol.

# CHAPTER II

# UC ZERO-KNOWLEDGE PROTOCOLS

In this chapter, we propose a complete security proof for the zero-knowledge proof based on signature cards of Hofheinz *et al.* [18] and present a new zero-knowledge protocol in the universal composable model using an idealized $TPM$ as catalyst. This chapter contains two sections. In Section 2.1, we introduce some of the definitions used in this work. We present protocol $SC\text{-}ZK^R$ and also a complete security proof. In Section 2.2, we introduce functionality $\mathcal{F}_{\mathcal{TPM}}^{\mathcal{T}}$, propose a protocol implementing functionality $\mathcal{F}_{ZK}^{R}$ and present a complete security proof for the TPM-based protocol.

## 2.1 Zero knowledge protocols based on Signature Cards

In this section, we present proof for universally composable security of the zero-knowledge protocol based on signature cards of Hofheinz *et al.* [18]. We list some notations and definitions used in rest of this work are below.

(1) **k** is a security parameter which controls the length of various parameters. In all cases in which lengths are a function of $k$, such a $n(k)$, $m(k)$, and $s(k)$, the function is polynomial in $k$.

(2) $\boldsymbol{w}$: a witness $w$ known to prover $P$ where $w \in \{0,1\}^{n(k)}$.

(3) **N**: A random nonce $N$ generated by verifier $V$ where $N \in \{0,1\}^{k}$.

(4) **r**: An arbitrary string $r$ generated by a party where $r \in \{0,1\}^{n(k)}$.

(5) **R**: A relation $R$ is defined as an $\mathcal{NP}$ relation where $R \subseteq \{0,1\}^{m(k)} \times \{0,1\}^{n(k)}$.

Recall that $x \in \{0,1\}^{m(k)}$ is the common input to both parties, i.e., honest prover $P$ and honest verifier $V$ in zero-knowledge protocols. $P$ knows a witness $w$ such that $xRw$. In developing their signature card assisted zero-knowledge protocols, Hofheinz *et al.* used a non-standard definition of zero-knowledge functionality that required transmitting an executable predicate to specify the relation to the ideal functionality [18]. While this allows a single functionality to serve arbitrary relations, it suffers from complications due to treating a relation as a transmittable parameter. In this thesis, we return to the traditional definitions of zero knowledge functionality in which the functionality is parametrized by the relation. The following definition is based on the original zero knowledge functionality defined by Goldwasser *et al.* [13]. The ideal functionality $\mathcal{F}_{\mathcal{ZK}}^{R}$ with relation $R$ is defined as follows

**Definition II.1.** $\mathcal{F}_{\mathcal{ZK}}^{R}$

**Functionality $\mathcal{F}_{\mathcal{ZK}}^{R}$**

$\mathcal{F}_{\mathcal{ZK}}^{R}$ proceeds as follows, running with a prover $P$, verifier $V$, and an adversary $\mathcal{S}$.

Upon receipt of an input (*prove*,*sid*,*x*,*w*) with $xRw$ from party $P$, send $(prove, sid, x)$ to $\mathcal{S}$. As soon as $\mathcal{S}$ allows the delivery, send $(proven, sid, x)$ to V where *sid* is the "session id" that is commonly used in universally composable definitions to distinguish between multiple uses of the same functionality.

Hofheinz's signature card based catalyst functionality $\mathcal{F}_{SigCard}^{\mathcal{G}}$ is given below.

**Functionality $\mathcal{F}_{SigCard}^{\mathcal{G}}$** (from [18])

For a signature scheme $\mathcal{G}$, $\mathcal{F}_{SigCard}^{\mathcal{G}}$ proceeds as follows, running with parties $P_1$,...,$P_n$ and an adversary $\mathcal{S}$.

**Initialization:** For each party $P_i$, generate a public/secret key $(pk_i, sk_i)$ and set $possessor_i := \perp$.

**Get public key:** When receiving a message $(getkey, sid, j)$ from some sub party $P$, send the public key $pk_j$ to $P$. Before delivering the key, ask the adversary (non-immediate delivery).

**Signature Generation:** Upon receiving a message $(sign, sid, m)$ from some subparty $P$ of $P_i$, if $P = possessor_i$, generate a signature $\sigma$ using $sk_i$, store the tuple $(i, m, \sigma)$, and send $(signature, sid, m, \sigma)$ to $P_i$.

**Signature Verification:** Upon receiving a message $(verify, sid, P_i, m, \sigma)$ from $P_j$ do: If a tuple $(i, m, \sigma)$ is a stored set $f = 1$ else set $f = 0$. Then if the public key of $P_i$ was already delivered to $P_j$ in some prior "Get public key" step, send $(verified, sid, m, f)$ to $P_j$.

**Possession:** Upon receiving a message $(seize, sid)$ from subparty $P$ (where $P$ is a subparty of $P_i$ or $P = \mathcal{S}$), if $possessor_i := \perp$, set $possessor_i := P$ and send $(seized, sid)$ to subparty $P$. Otherwise send $(occupied, sid)$ to subparty $P$.

**Dispossession:**Upon receiving a message $(release, sid)$ from subparty $P$ (where $P$ is a subparty of $P_i$ of $P = \mathcal{S}$), if $possessor_i := P$, set $possessor_i := \perp$.

The functionalities of the ideal functionality $\mathcal{F}_{SigCard}^{\mathcal{G}}$ include generating signatures. We list definitions of some signatures generated by $\mathcal{F}_{SigCard}^{\mathcal{G}}$ and some related

notations which are used in rest of this work here.

**Definition II.2. Signatures by Signature Cards**

(1) $\boldsymbol{s_N}$: Signature generated by $\mathcal{F}_{SigCard}^{\mathcal{G}}$ functionality of verifier $V$ on a random string $N$ where $s_N \in \{0,1\}^{s(k)}$.

(2) $\boldsymbol{s_w}$: Signature generated by $\mathcal{F}_{SigCard}^{\mathcal{G}}$ functionality of prover $P$ on a witness $w$ where $s_w \in \{0,1\}^{s(k)}$.

(3) $\boldsymbol{s_R}$: An arbitrary string generated by a party with length same as the signature generated by $\mathcal{F}_{SigCard}^{\mathcal{G}}$ where $s_R \in \{0,1\}^{s(k)}$.

*Protocol SC-ZK$^R$*

We present the protocol *SC-ZK$^R$* of Hofheinz *et al.*[18], with modified notations which reflect the use of the traditional zero knowledge functionality. Participants in protocol *SC-ZK$^R$* are prover $P$, verifier $V$, real adversary $\mathcal{A}$, ideal simulator $\mathcal{S}$ and the environment $\mathcal{Z}$. All communication is done through a secure channel that only leaks the length of the messages. $\mathcal{Z}$ may access the functionality $\mathcal{F}_{SigCard}^{\mathcal{G}}$ through parties other than $P$ and $V$. $P$ knows $(x, w)$ such that $xRw$.

(1) When receiving an input $(prove, sid, x, w)$, where $x$ is the common input for $R$ such that $xRw$, $P$ sends $x \in \{0,1\}^{m(k)}$ to $V$.

(2) $V$ seizes its signature card. If it cannot seize the card, it terminates.

(3) $V$ generates a random nonce $N$ of $k$ bits. This nonce is sent via a secure channel to $P$.

(4) $P$ requests the public keys $pk_V$ and $pk_P$ of $V$ and $P$, respectively, from $\mathcal{F}_{SigCard}^{\mathcal{G}}$. Then it seizes its signature card, signs $w$ and releases its signature card. Next, it proves to the $V$ that it knows a triple $(w, s_w, s_R)$ such that $verify_{pk_P}(1^k, s_w, w) \wedge xRw \vee verify_{pk_V}(1^k, s_R, N)$ using a witness indistinguishable argument of knowledge (WIAOK).

(5) If $V$ accepts the argument of knowledge, it terminates with output $(proven, sid, x)$. Additionally, it releases its signature card in any case.

**Theorem II.3.** *If $\mathcal{G}$ is an existentially unforgeable signature scheme, then protocol SC-ZK$^R$ using the functionality $\mathcal{F}_{SigCard}^{\mathcal{G}}$ securely implements the functionalities $\mathcal{F}_{ZK}^{R}$ and $\mathcal{F}_{SigCard}^{\mathcal{G}}$ with respect to static adversaries. Here $\mathcal{F}_{SigCard}^{\mathcal{G}}$ is used as a catalyst.*

*Proof.* In order to prove the security of *SC-ZK*$^R$, we consider two scenarios of implementing protocol *SC-ZK*$^R$, one with $V$ corrupted and the other with $P$ corrupted. In both cases we consider two settings: one in which the real protocol runs with real parties, and another with a simulator $\mathcal{S}$ interacting with ideal functionality $\mathcal{F}_{ZK}^{R}$. At the finish of either setting, the environment provides an output $\mathcal{Z}(ensemble)$, which is a random variable (i.e., the output of $\mathcal{Z}$ on input *ensemble*).

*Case 1 : Corrupted Verifier*

Two setting are described below.

*Real adversary $\mathcal{A}$*

As the verifier $V$ is corrupted, the protocol *SC-ZK*$^R$ will be modified. Adversary $\mathcal{A}$ takes control of the corrupted verifier $V$ and stands in for $V$ in *SC-ZK*$^R$. Step 3 of the above protocol *SC-ZK*$^R$ will be modified where $\mathcal{A}$ will generate nonce $N$ of $k$ bits and send it to the $P$. And, step 5 also be modified, as $\mathcal{A}$ will be responding

to the $P$'s WIAOK and delivering the output to the environment $\mathcal{Z}$. After step 3 of *SC-ZK*$^R$, both parties will have same input $I = \langle x, N \rangle \in \{0,1\}^{m(k)} \times \{0,1\}^k$. The witness triple for the WIAOK will be $W_A = (w, s_w, s_R)$, and as honest prover $P$ knows the witness $w$ and its signature $s_w$, it will generate WIAOK for $verify_{pk_p}(1^k, s_w, w) \wedge xRw \vee verify_{pk_v}(1^k, s_R, N)$. The ensemble generated by $\mathcal{A}$, on interaction with honest $P$ with witness $W_A$ on common input $I$ is $\{\langle P(W_A), \mathcal{A} \rangle (I)\}_{I \in \{0,1\}^{m(k)} \times \{0,1\}^k}$.

*Ideal adversary* $\mathcal{S}$

As the verifier is corrupted, simulator $\mathcal{S}$ simulates the static adversary $\mathcal{A}$ with real parties, i.e., with honest prover $P$ and corrupted verifier $V$. When the $\mathcal{F}^R_{\mathcal{ZK}}$ receives (*prove,sid,x,w*) from party $P$, and if $xRw$, then $\mathcal{F}^R_{\mathcal{ZK}}$ sends (*prove,sid,* $x$) to $\mathcal{S}$. Note that while $P$ knows $w$, simulator $\mathcal{S}$ is unaware of $w$ inorder to prove $xRw$. As soon as $\mathcal{S}$ receives (*prove,sid,* $x$) it seizes $V$'s signature card and simulates real adversary $\mathcal{A}$ with real corrupted verifier $V$ from step 3 of protocol *SC-ZK*$^R$. After the simulation of $\mathcal{A}$ produces nonce $N$, $\mathcal{S}$ sends $N$ to $V$'s signature card and receives $s_N$ from $V$'s signature card. Next, $\mathcal{S}$ sets up a WIAOK using the common input $I = \langle x, N \rangle \in \{0,1\}^{m(k)} \times \{0,1\}^k$ and witness triple $W_S = (r, s_R, s_N)$, and the WIAOK will argue that $verify_{pk_p}(1^k, s_R, r) \wedge xRw \vee verify_{pk_v}(1^k, s_N, N)$. The ensemble generated by $\mathcal{S}$ in this case will be $\{\langle \mathcal{S}(W_S), \mathcal{S} \rangle (I)\}_{I:\{0,1\}^{m(k)} \times \{0,1\}^k}$.

In the two settings, real and ideal, the behavior seen by the environment $\mathcal{Z}$ is identical through step 3 of *SC-ZK*$^R$. The only difference is in the WIAOK in step 4 and step 5, so in effect environment $\mathcal{Z}$ acts as a polynomial-time distinguisher for the real and ideal settings. If $\mathcal{Z}$ can distinguish between the real world adversary $\mathcal{A}$ and ideal adversary $\mathcal{S}$ with non negligible probability $\frac{1}{p(k)}$ , then the above construction acts as

a distinguisher for the WIAOK witnesses $W_A$ and $W_S$ with non-negligible probability, violating our assumption of secure WIAOK. Therefore, it follows for any polynomial $p(k)$ and all sufficiently large $k$ it holds

$Pr[\mathcal{Z}(I, \langle P(W_A), \mathcal{A}\rangle(I)) = 1] - Pr[\mathcal{Z}(I, \langle P(W_S), \mathcal{S}\rangle(I)) = 1] < \frac{1}{p(k)}$.

*Case 2: Corrupted Prover*

We consider two settings, i.e., real adversary $\mathcal{A}$ interacting with *SC-ZK*$^R$ and ideal adversary $\mathcal{S}$ interacting with $\mathcal{F}^R_{\mathcal{ZK}}$. In both settings, an honest $V$ will be participating. The sequence of steps of $V$'s interaction with *SC-ZK*$^R$ plays a vital role in the proof. $V$ seizes its signature card (in step 2) and then generates a random nonce $N$. $V$ releases its signature only at the end of *SC-ZK*$^R$ (step 5). This prevents any other party from seizing $V$'s signature card in the course of *SC-ZK*$^R$ (step 2 through step 5) to generate a fake signature $s_N$ for nonce $N$ and generate the WIAOK.

*Real Adversary* $\mathcal{A}$

As prover $P$ is corrupted, adversary $\mathcal{A}$ takes control of prover $P$ which leads to the modification of *SC-ZK*$^R$. Step 1 of *SC-ZK*$^R$ will be modified, where $\mathcal{A}$ sends $x \in \{0,1\}^{m(k)}$ on behalf of $P$. Step 4 of *SC-ZK*$^R$ will be modified, where $\mathcal{A}$ will prove to the verifier that there exists a triple $(w, s_w, s_R)$ such that $\mathit{verify}_{pk_P}(1^k, s_w, w) \wedge xRw \vee \mathit{verify}_{pk_V}(1^k, s_R, N)$ using a WIAOK. After step 3 of modified protocol *SC-ZK*$^R$, both parties adversary $\mathcal{A}$ and honest verifier $V$ will have the same input $I = \langle x, N\rangle \in \{0,1\}^{m(k)} \times \{0,1\}^k$ The witness triple in this case is $W_A = (w, s_w, s_R)$ and WIAOK is $\mathit{verify}_{pk_p}(1^k, s_w, w) \wedge xRw \vee \mathit{verify}_{pk_v}(1^k, s_R, N)$. The ensemble generated by honest $V$, on interaction with corrupted $P$ with witness $W_A$ on common input $I$

is $\{\langle \mathcal{A}(W_A), V\rangle(I)\}_{I\in\{0,1\}^{m(k)}\times\{0,1\}^k}$

*Ideal Adversary* $\mathcal{S}$

As prover $P$ is corrupted, simulator $\mathcal{S}$ simulates the static adversary $\mathcal{A}$ with real parties, i.e., corrupted prover $P$ and honest verifier $V$. As a part of simulation of the real protocol *SC-ZK*$^R$ with static adversary $\mathcal{A}$, step 1 of *TPM-ZK*$^R$ will be modified where $\mathcal{A}$ stands in for corrupted $P$. In step 1, $\mathcal{A}$ will send $x \in \{0,1\}^{m(k)}$ to honest verifier $V$. Step 4 will be modified where $\mathcal{S}$ seizes corrupted $P$'s signature card and $\mathcal{A}$ sends $w$ to $\mathcal{S}$ to get $s_w$. Since the signature card is a catalyst, $\mathcal{A}$ may send many other values to $\mathcal{S}$ (which seized $P$'s signature cards) for signatures of those values. In order to capture $w$, $\mathcal{S}$ tests every value from $\mathcal{A}$ to see which value $w$ satisfies $xRw$ and then stores both $w$ and $s_w$. After generating $s_w$, $\mathcal{S}$ delivers $s_w$ to $\mathcal{A}$ and releases $P$'s signature card. In the ideal world, $\mathcal{S}$ acts as corrupted $P$ and sends $(prove, sid, x, w)$ to ideal functionality $\mathcal{F}_{ZK}^R$. Since $xRw$, then $\mathcal{F}_{ZK}^R$ sends $(prove, sid, x)$ to $\mathcal{S}$. Another modification for step 4 is $\mathcal{A}$ will prove to the honest verifier $V$ of $\mathcal{F}_{\mathcal{ZK}}^R$ that there exists a triple $(w, s_w, s_R)$ such that $verify_{pk_p}(1^k, s_w, w) \wedge xRw \vee verify_{pk_v}(1^k, s_R, N)$ using a WIAOK. Depending on real honest verifier $V$'s output at step 5, the simulator $\mathcal{S}$ either allows or does not allow the ideal functionality $\mathcal{F}_{\mathcal{ZK}}^R$ to send (*proven*,*sid*,$x$) to ideal party $V$. Since simulator $\mathcal{S}$ simulates the complete modified protocol *SC-ZK*$^R$ with static adversary $\mathcal{A}$, the ensemble generated by the real honest verifier is $\{\langle \mathcal{A}(W_A), V\rangle(I)\}_{I\in\{0,1\}^{m(k)}\times\{0,1\}^k}$, where $I$ is the common input. Finally, the environment $\mathcal{Z}$ cannot distinguish with which adversary it is interacting, because in both scenarios the honest verifier produces indistinguishable ensembles.

Therefore, from both cases we have shown that protocol *SC-ZK*$^R$ using the functionality $\mathcal{F}_{SigCard}^{\mathcal{G}}$ securely implements the functionalities $\mathcal{F}_{\mathcal{ZK}}^R$ and $\mathcal{F}_{SigCard}^{\mathcal{G}}$ with re-

spect to static adversaries. □

## 2.2 Zero-knowledge protocols based on TPMs

In this section, we propose a zero-knowledge protocol in the universal composable model of security, using a tamper proof device called a Trusted Platform Module as a catalyst. As described earlier in this chapter, Hofheinz *et al.*[18] use signature cards as catalysts in designing both zero-knowledge proofs and commitments in universal composable settings. In this section we use Trusted Platform Modules (TPMs) as a catalyst in designing a zero-knowledge protocol with universally composable security.

$\mathcal{F}_{TPM}^{\mathcal{T}}$

The catalyst functionality in this protocol is $\mathcal{F}_{TPM}^{\mathcal{T}}$, which we given below

**Functionality $\mathcal{F}_{TPM}^{\mathcal{T}}$**

**Part I**

For a signature scheme $\mathcal{T}$, $\mathcal{F}_{TPM}^{\mathcal{T}}$ proceeds as follows,running with parties $P_1$,...,$P_n$ and an adversary $\mathcal{S}$.

**Initialization:** For each party $P_i$, a $TPM_i$ with a legitimate Endorsement Key (EK) is given.

**Create AIK:** When receiving a message $(createaik, sid)$ from some sub party $P$ of $P_i$, creates an Attestation Identity Key $AIK_i$, generates certificate $cert_{AIK_i}$ from Trusted CA using makeidentity functionality, stores a tuple $(sid, i, AIK_i, cert_{AIK_i})$ and returns $(pub_{AIK_i}, cert_{AIK_i}, sid)$ to $P_i$.

**Functionality $\mathcal{F}_{TPM}^{\mathcal{I}}$**

**Part II**

**Create Key:** When receiving a message $(createkey, sid, secret)$ from some sub party $P$ of $P_i$, creates a signing key $key_i$. If there exists a stored tuple $(sid, i, AIK_i, cert_{AIK_i})$, then generate certificate $cert_{key_i}$ for $key_i$ using $AIK_i$ with the certifykey functionality of TPM. Stores tuple $(cert_{key_i}, key_i, secret)$ and returns $(pub_{key_i}, cert_{key_i}, sid)$ to $P_i$.

**Load Key:** When receiving a message $(loadkey, sid, key_i, secret)$ from some sub party $P$ of $P_i$, $TPM_i$ returns $keyhandle_i$ which refers to internally loaded private key $key_i$ into $TPM_i$. Next, stores a tuple $(keyhandle_i, key_i, sid, i)$ and returns $(keyhandle_i, sid)$ to $P_i$.

**Get public key:** When receiving a message $(getkey, sid, cert_{key_j}, j)$ from some sub party $P$. If there exists a stored tuples $(sid, j, AIK_j, cert_{AIK_j})$ and $(cert_{key_j}, key_j)$ then it gets both $key_j$ and $cert_{AIK_j}$ and sends the public key tuple $(key_j, cert_{AIK_j})$ to P. Before delivering the tuple with key, ask the adversary (non-immediate delivery).

**Signature Generation:** Upon receiving a message $(sign, sid, m, keyhandle_i)$ from some subparty $P$ of $P_i$, if there exists a stored tuple $(keyhandle_i, key_i, sid, i)$, then generate a signature $\sigma$ using $key_i$, store the $tuple(i, m, \sigma, key_i)$ and send $(signature, sid, m, \sigma)$ to $P_i$.

**Signature Verification:** Upon receiving a message $(verify, sid, P_i, m, \sigma, key_i)$ from $P_j$ do: If a tuple $(i, m, \sigma, key_i)$ is stored set $f = 1$ else set $f = 0$. Sends $(verified, sid, m, f)$ to $P_j$.

**Functionality $\mathcal{F}_{TPM}^{\mathcal{T}}$**

**Part III**

**Unload Key:** Upon receiving a message $(unload, sid, keyhandle_i)$ from subparty $P$ (where $P$ is a subparty of $P_i$ or $P = \mathcal{S}$), if there exists tuple $(keyhandle_i, key_i, sid, i)$, then discard the stored tuple $(keyhandle_i, key_i, sid, i)$.

As the functionalities of ideal functionality $\mathcal{F}_{TPM}^{\mathcal{T}}$ include generating signatures, we list definitions of some signatures generated by $\mathcal{F}_{TPM}^{\mathcal{T}}$ and some related notations which are used in the rest of this work below.

**Definition II.4. Signatures by Trusted Platform Modules**

(1) $\boldsymbol{t_N}$: Signature generated by $\mathcal{F}_{TPM}^{\mathcal{T}}$ functionality of verifier $V$'s TPM using its signing key $key_V$ on a random string $N$ where $t_N \in \{0,1\}^{s(k)}$.

(2) $\boldsymbol{t_w}$: Signature generated by $\mathcal{F}_{TPM}^{\mathcal{T}}$ functionality of prover $P$'s TPM using its signing key $key_V$ on a witness $w$ where $t_w \in \{0,1\}^{s(k)}$.

(3) $\boldsymbol{t_R}$: An arbitrary string generated by a party with length same as the signature generated by $\mathcal{F}_{TPM}^{\mathcal{T}}$ where $t_R \in \{0,1\}^{s(k)}$.

*Protocol TPM-ZK$^R$*

Besides using a TPM as a catalyst, we use witness indistinguishable arguments (WIAOK) in a way similar to the protocol of introduced in Hofheinz *et al.*[18] for the protocol *TPM-ZK$^R$*. The participants of *TPM-ZK$^R$* are prover $P$, verifier $V$, real

adversary $\mathcal{A}$, simulator $\mathcal{S}$ and environment $\mathcal{Z}$. *TPM-ZK*$^R$ implements functionality $\mathcal{F}_{ZK}^R$ using $\mathcal{F}_{TPM}^{\mathcal{T}}$ as catalyst, where all communication is done through a secure channel that only leaks the length of the messages, and the environment may access the functionality $\mathcal{F}_{TPM}^{\mathcal{T}}$ through parties other than $P$ and $V$. At the beginning of the protocol *TPM-ZK*$^R$, $P$ knows $(x, w)$ such that $xRw$.

(1) When receiving an input $(prove, sid, x, w)$, where $x$ is the common input for $R$ such that $xRw$, the prover $P$ sends $x$ to $V$.

(2) $V$ creates a signing key $key_V$ by sending $(createaik, sid, secret)$ to $\mathcal{F}_{TPM}^{\mathcal{T}}$ and loads $key_V$ by sending $(loadkey, sid, key_i, secret)$ to $\mathcal{F}_{TPM}^{\mathcal{T}}$ functionality. If it cannot load the $key_V$, it terminates.

(3) $V$ generates a random nonce $N$ of $k$ bits and sends tuple $(N, cert_{key_V})$ to $P$ via a secure channel.

(4) $P$ gets the public key tuple of $V$ using get key functionality of $\mathcal{F}_{TPM}^{\mathcal{T}}$ functionality. Then, $P$ creates a signing key $key_P$ and loads $key_P$ into its $TPM_P$ using create key and load key fuctionalities of $\mathcal{F}_{TPM}^R$ functionality respectively. If it cannot load the $key_P$, it terminates. $P$ signs $w$ by sending $(sign, sid, m, keyhandle_i)$ to $\mathcal{F}_{TPM}^{\mathcal{T}}$, and unloads its key $key_P$ using the unload key functionality of $\mathcal{F}_{TPM}^{\mathcal{T}}$. Then it proves to the verifier that there exists a triple $(w, t_w, t_N)$, such that $verify_{key_P}(1_k, t_w, w) \vee xRw \vee verify_{key_V}(1^k, t_N, N)$ using a witness indistinguishable argument of knowledge (WIAOK). Along with WIAOK, $P$ sends $cert_{key_P}$ to $V$.

(5) If the verifier accepts the arguments of knowledge, it terminates with output

$(proven, sid, x)$. Additionally, it unloads its key in any case.

**Theorem II.5.** *If $\mathcal{T}$ is an existentially unforgeable signature scheme, then protocol TPM-ZK$^R$ using the functionality $\mathcal{F}_{TPM}^{\mathcal{T}}$ securely implements the functionalities $\mathcal{F}_{ZK}^{R}$ and $\mathcal{F}_{TPM}^{\mathcal{T}}$ with respect to static adversaries. Here $\mathcal{F}_{TPM}^{\mathcal{T}}$ is used as a catalyst.*

*Proof.* The participants of protocol *TPM-ZK$^R$* are environment $\mathcal{Z}$, prover $P$, verifier $V$, real adversaries $\mathcal{A}$ and ideal adversary $\mathcal{S}$. In order to prove the security, we consider two scenarios of protocol *TPM-ZK$^R$*, where prover $P$ is corrupted in one scenario and verifier $V$ is corrupted in other. In each scenario, we consider two settings: one in which the real protocol is run with real parties, and another with a simulator $\mathcal{S}$ interacting with ideal functionality $\mathcal{F}_{ZK}^{R}$. At the finish of either setting of each scenario, the environment provides an output $\mathcal{Z}(ensemble)$, which is a random variable (i.e., the output of $\mathcal{Z}$ on input *ensemble*).

*Case 1: Corrupted Verifier*

Two settings are as follows

*Real Adversary $\mathcal{A}$*

The protocol *TPM-ZK$^R$* will be modified, where $\mathcal{A}$ stands in for $V$ in *TPM-ZK$^R$*. In step 3, $\mathcal{A}$ will generate nonce $N$ of $k$ bits and send it to $P$. At step 4, adversary $\mathcal{A}$ will be responding to the prover $P$'s WIAOK and delivers the output to the environment $\mathcal{Z}$. After step 3 of *TPM-ZK$^R$*, both parties will have same input $I = \langle x, N \rangle \in \{0,1\}^{m(k)} \times \{0,1\}^k$. The witness triple known to $P$ will be $W_A = (w, t_w, t_R)$, and the honest prover $P$ will generate a WIAOK for $verify_{key_P}(1^k, t_w, w) \wedge$

$xRw \vee verify_{key_V}(1^k, t_R, N)$. The ensemble generated by $\mathcal{A}$, on interaction with honest $P$ with witness $W_A$ on common input $I$ is $\{\langle P(W_A), \mathcal{A}\rangle(I)\}_{I \in \{0,1\}^{m(k)} \times \{0,1\}^k}$.

*Ideal adversary* $\mathcal{S}$

$\mathcal{S}$ simulates the static adversary $\mathcal{A}$ with real parties, i.e., with honest prover $P$ and corrupted verifier $V$. When $\mathcal{F}_{\mathcal{ZK}}^R$ receives (*prove,sid,x,w*) from party $P$, and if $xRw$, then $\mathcal{F}_{\mathcal{ZK}}^R$ sends (*prove,sid*, $x$) to $\mathcal{S}$. As soon as $\mathcal{S}$ receives $(prove, sid, x)$ it initializes $V$'s $\mathcal{F}_{TPM}^{\mathcal{T}}$ functionality, uses $\mathcal{F}_{TPM}^{\mathcal{T}}$ to generate a key $key_j$ and simulates protocol *TPM-ZK*$^R$ with adversary $\mathcal{A}$. The real protocol *TPM-ZK*$^R$ will be modified from step 3 where $\mathcal{A}$ stands in for corrupted $V$. $\mathcal{A}$ produces nonce $N$ and sends to $\mathcal{S}$. $\mathcal{S}$ sends $N$ to $V$'s $\mathcal{F}_{TPM}^{\mathcal{T}}$ and receives $t_N$ from $V$'s $\mathcal{F}_{TPM}^{\mathcal{T}}$. Next, $\mathcal{S}$ sets up a WIAOK using common input is $I = \langle x, N\rangle \in \{0,1\}^{m(k)} \times \{0,1\}^k$ and witness triple $W_S = (r, t_R, t_N)$, and the WIAOK will argue that $verify_{key_P}(1^k, t_R, r) \wedge xRw \vee verify_{key_V}(1^k, t_N, N)$. The ensemble generated by $\mathcal{S}$ in this case will be $\{\langle \mathcal{S}(W_S), \mathcal{S}\rangle(I)\}_{I \in \{0,1\}^{m(k)} \times \{0,1\}^k}$

In two settings, real and ideal, the behavior seen by the environment $\mathcal{Z}$ is identical through step 3 of *TPM-ZK*$^R$. The only difference is in the WIAOK in steps 4 and step 5, so in effect the environment $\mathcal{Z}$ acts as a polynomial-time distinguisher for the real and ideal settings. If $\mathcal{Z}$ can distinguish between the real world adversary $\mathcal{A}$ and ideal adversary $\mathcal{S}$ with non negligible probability $\frac{1}{p(k)}$, then the above construction acts as a distinguisher for the WIAOK witnesses $W_A$ and $W_S$ with non-negligible probability, violating our assumption of secure WIAOK. Therefore, it follows for any polynomial $p(k)$ and all sufficiently large $k$ and it holds

$Pr[\mathcal{Z}(I, \langle P(W_A), \mathcal{A}\rangle(I)) = 1] - Pr[\mathcal{Z}(I, \langle P(W_S), \mathcal{S}\rangle(I)) = 1] < \frac{1}{p(k)}$

*Case 2: Corrupted Prover*

In both settings, honest $V$ will be participating in *TPM-ZK*$^R$. The two settings are:

*Real Adversary* $\mathcal{A}$

As $P$ is corrupted, $\mathcal{A}$ stands in for corrupted $P$ which leads to the modification of *TPM-ZK*$^R$. Step 1 of *TPM-ZK*$^R$ will be modified, where $\mathcal{A}$ sends $x \in \{0,1\}^{m(k)}$ to honest $V$. In step 4, $\mathcal{A}$ will prove to the verifier that there exists a triple $(w, t_w, t_R)$ such that $verify_{key_P}(1^k, t_w, w) \wedge xRw \vee verify_{key_V}(1^k, t_R, N)$ using a WIAOK. After step 3 of modified protocol *TPM-ZK*$^R$, both parties $\mathcal{A}$ and honest $V$ will have the same input $I = \langle x, N \rangle \in \{0,1\}^{m(k)} \times \{0,1\}^k$. The witness triple in this case is $W_A = (w, t_w, t_R)$ and WIAOK is $verify_{key_P}(1^k, t_w, w) \wedge xRw \vee verify_{key_V}(1^k, t_R, N)$. The ensemble generated by honest $V$, on interaction with corrupted $P$ with witness $W_A$ on common input $I$ is $\{\langle \mathcal{A}(W_A), V \rangle(I)\}_{I \in \{0,1\}^{m(k)} \times \{0,1\}^k}$.

*Ideal Adversary* $\mathcal{S}$

As prover $P$ is corrupted, simulator $\mathcal{S}$ simulates *TPM-ZK*$^R$ with static adversary $\mathcal{A}$ and real parties, i.e., corrupted prover $P$ and honest verifier $V$. As a part of simulation of the real protocol *TPM-ZK*$^R$ with static adversary $\mathcal{A}$, step 1 of *TPM-ZK*$^R$ will be modified because $\mathcal{A}$ stands in for corrupted $P$. In step 1, $\mathcal{A}$ will send $x \in \{0,1\}^{m(k)}$ to honest verifier $V$. In step 4, $\mathcal{S}$ initialize corrupted $P$'s $\mathcal{F}_{TPM}^{\mathcal{T}}$ and $\mathcal{A}$ sends $w$ to $\mathcal{S}$ to get $t_w$. Since TPM is a catalyst, $\mathcal{A}$ may send many other values to $\mathcal{S}$ (which controls the $P$'s TPM) for signatures of those values. In order to capture $w$, $\mathcal{S}$ tests every value from $\mathcal{A}$ to find one which satisfies $xRw$, and then stores

both $w$ and $s_w$. After generating $t_w$, $\mathcal{S}$ delivers $t_w$ to $\mathcal{A}$ and unload $key_P$ from $P$'s TPM. In the ideal world, $\mathcal{S}$ acts as corrupted $P$ and sends $(prove, sid, x, w)$ to ideal functionality $\mathcal{F}_{ZK}^{R}$. If $xRw$, then $\mathcal{F}_{ZK}^{R}$ sends $(prove, sid, x)$ to $\mathcal{S}$. Another modification for step 4 is $\mathcal{A}$ will prove to the honest verifier $V$ of $\mathcal{F}_{\mathcal{ZK}}^{R}$ that there exists a triple $(w, t_w, t_R)$ such that $verify_{pk_p}(1^k, s_w, w) \wedge xRw \vee verify_{pk_v}(1^k, s_R, N)$ using a WIAOK. Depending on real honest verifier $V$'s output at step 5, the simulator $\mathcal{S}$ either allows or does not allow the ideal functionality $\mathcal{F}_{\mathcal{ZK}}^{R}$ to send $(proven,sid,x)$ to ideal party $V$. Since simulator $\mathcal{S}$ simulates the complete modified protocol $TPM\text{-}ZK^R$ with static adversary $\mathcal{A}$, the ensemble generated by the real honest verifier is $verify_{key_P}(1^k, t_w, w) \wedge xRw \vee verify_{key_V}(1^k, t_R, N)$, where $I$ is the common input. Finally, the environment $\mathcal{Z}$ cannot distinguish with which adversary it is interacting because in both scenarios the honest verifier produces indistinguishable ensembles.

Therefore, from both cases we have shown that protocol $TPM\text{-}ZK^R$ using the functionality $\mathcal{F}_{TPM}^{\mathcal{T}}$ securely implements the functionalities $\mathcal{F}_{\mathcal{ZK}}^{R}$ and $\mathcal{F}_{TPM}^{\mathcal{T}}$ with respect to static adversaries. □

# CHAPTER III

# CONCLUSION AND FUTURE DIRECTIONS

The idea of using actual secure hardware in designing universally composable cryptographic protocols, pursued by Hofheinz *et al.* [18], is an important breakthrough in designing realistic two party protocols in universal composable framework. Hofheinz *et al.* use government issued signature cards as *catalysts* to support universally composable zero knowledge and commitments protocols. The ideal functionality which is securely realized by zero-knowledge protocol of Hofheinz *et al.* is not the standard definition. Therefore, in this thesis we returned to the traditional definition of the ideal functionality using a generic witness relation, which reflects the Canneti's ideal functionality [3]. While providing a brief sketch, Hofheinz *et al.* did not give a complete security proof for the universally composable zero-knowledge proofs from signature cards. As one contribution of this thesis, we present a complete security proof for this universally composable zero-knowledge protocol, using signature cards and based on a generic witness relation.

Some of the drawbacks of relying on these government issued signature cards of Hofheinz *et al.* [18] are:

(1) Although, signature cards are tamper evident, they are not widespread.

(2) The parties in the protocol must completely trust the producer or issuer of the signature cards. If the issuer is corrupted, then whole setting will be meaningless.

(3) Since the signature cards are government issued, in a scenario in which parties from different countries participate in a protocol, the system using signature cards may become more complex or impractical.

These drawbacks can be mitigated or completely eliminated by using Trusted Platform Modules (TPMs), tamper-evident chips designed to be embedded in computer systems. Trusted Platform Modules are widely spread and are attached to the motherboard of almost every business laptops or desktops currently produced. Since they are widespread, it gives us flexibility to design realistic universally composable multi-party protocols using Trusted Platform Modules. In this thesis, we give a detailed definition of ideal functionality of a TPM-based catalyst, and present a universally composable zero knowledge protocol using Trusted Platform Modules (TPMs). We also provide a complete security proof for our universally composable zero knowledge protocol using a TPM-based catalyst.

There are several possible future directions to extend this work. First, our proposed TPM-assisted universally composable zero knowledge protocol creates a new signing key for every instance of its execution. Since a TPM is a relatively slow device, creating a key is a costly operation. Improvements should be made in the design of protocol to address this issue effectively. Additionally, in the current work we require every party have a TPM, but the TPMs of the two parties play different roles. Therefore, another direction is to consider whether a secure protocol can be designed with a TPM required by only one party of the protocol.

## REFERENCES


[1] Backes, M., Hofheinz, D., Muller-Quade, J., and Unruh, D. On fairness in simulatability-based cryptographic systems. In *Proceedings of the 3rd ACM Workshop on Formal Methods in Security Engineering (FMSE)* (2005), pp. 13–22.

[2] Barak, B., Canetti, R., Lindell, Y., Pass, R., and Rabin, T. Secure computation without authentication. In *CRYPTO'05* (2005), pp. 361–377.

[3] Canetti, R. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (2001), pp. 136–145.

[4] Canetti, R., Dodis, Y., Pass, R., and Walfish, S. Universal composable security with global setup. In *Proceedings of 4th Theory of Cryptography Conference (TCC'07)* (2007), pp. 61–85.

[5] Canetti, R., and Fischlin, M. Universally composable commitments. In *CRYPTO'01* (2001), pp. 19–40.

[6] Canetti, R., Kushilevitz, E., and Lindell, Y. On the limitations of universally composable two-party computation without set-up assumptions. In *CRYPTO'03* (2003), pp. 135–167.

[7] Canneti, R., Kushilevitz, E., and Lindell, Y. On the limitations of universally composable two-party computation without set-up assumptions. In *EUROCRYPT'03* (2003), pp. 68–86.

[8] Chandran, N., Goyal, V., and Sahai, A. New Constructions for UC Secure Computation using Tamper Proof Hardware. In *EUROCRYPT'08* (2008), pp. 545–562.

[9] Damgard, I., Nielsen, J. B., and Wichs, D. Isolated proofs of knowledge and isolated zero knowledge. In *EUROCRYPT'08* (2008), pp. 509–526.

[10] Diffie, W., and Hellman, M. New directions in cryptography. *IEEE Transactions on Information Theory* (1977), 644–654.

[11] Feige, U., and Shamir, A. Witness indistinguishable and witness hiding protocols. In *Proceedings of 22nd ACM Symposium Theory of Computing (STOC'90)* (1990), pp. 416–426.

[12] GoldReich, O. *Foundations of Cryptography : Basic Tools.* Cambridge University Press, 2001.

[13] Goldwasser, S., Micali, S., and Rackoff, C. The knowledge complexity of interactive proof-systems. *Society for Industrial and Applied Mathematics (SIAM) Journal on Computing* (1989), 186–208.

[14] Goldwasser, S., and Sipser, M. Private coins versus public coins in interactive proof systems. In *Proceedings of ACM Symposium Theory of Computing (STOC'86)* (1986), pp. 58–68.

[15] Group, T. C. Trusted platform module specifications – parts 1–3. Available at https://www.trustedcomputinggroup.org/specs/TPM/.

[16] Gunupudi, V., and Tate, S. Random oracle instantiation in distributed protocols using trusted platform modules. In *21st International Confernece on Advanced Information Networking Applications* (2007), pp. 463–469.

[17] Hofheinz, D., and Muller-Quade, J. Universally composable commitments using random oracles. In *Proceedings of Theory of Cryptography Conference (TCC)* (2004), pp. 58–76.

[18] Hofheinz, D., Muller-Quade, J., and Unruh, D. Universally Composable Zero-knowledge Arguments and Commitments from Signature Cards. In *Proceedings of the 5th Central European Conference on Cryptology MoraviaCrypt* (2005).

[19] Katz, J. Universally composable multi-party computation using tamper-proof hardware. In *Advances in Cryptology - EUROCRYPT 2007* (2007), pp. 115–128.

[20] Moran, T., and Segev, G. David and Goliath Commitments: UC Computation for Asymmetric Parties using Tamper-proof Hardware. In *EUROCRYPT'08* (2008), pp. 527–544.

[21] Rivest, Shamir, and Adleman. A method for obtaining digital signatures and public-key cryptosystems. In *Communications of the ACM 21* (1978), pp. 120–126.

[22] Soanes, C., and Stevenson, A. *Oxford dictionary of English.* Oxford University Press, 2005.

[23] Tate, S., and Vishwanathan, R. Improving cut-and-choose in verifiable encryption and fair exchange protocols using trusted computing technology. In *Data and Applications Security (DBSec'09)* (2009), pp. 252–267.

[24] Tuchman, W. *A brief history of the data encryption standard.* ACM Press/Addison-Wesley Publishing, 1997.