Trust via Common Languages

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Ingy A.K. Ramzy Youssef, B.S., M.S.

Graduate Program in Department of Computer Science and Engineering

The Ohio State University

2016

Dissertation Committee:

Anish Arora, Advisor
Dong Xuan
Timothy Carlson
Yinqian Zhang
© Copyright by
Ingy A.K. Ramzy Youssef

2016
Abstract

To prove the authenticity of a node $X$ to a node $Y$ that does not know $X$, the conventional approach is to use $X$’s knowledge of some certified identity $I$. The use of knowledge at $Y$ is eschewed, or used only to model adversarial strategies, as opposed to improve the security properties of authentication protocols. In this thesis, we consider using the knowledge at $Y$ along with the knowledge at $X$ to provide an alternative basis of trust; instead of $X$ proving identifying knowledge to $Y$, both $X$ and $Y$ use their knowledge to prove knowledge commonalities, i.e., that their knowledge is related, which serves as a basis of trust between them.

Our approach to establishing trust between $X$ and $Y$ allows us to forgo the use of certified identities. We define relations between the knowledge of $X$ and $Y$ on common values, thereby constituting a common language $L$ of both. Interaction between $X$ and $Y$ on input $I$ yields a proof of membership of $I$ in $L$ allowing $X$ and $Y$ to trust each other on the basis of their related knowledge. To this end, it suffices to design the common language and its corresponding relation to be used as a basis of authentication instead of certified identities. Thus, while authentication has usually been at odds with privacy and anonymity, using proofs on common languages makes it possible to achieve authentication that is privacy and anonymity preserving.
We propose the common language model along with a new interactive proof system of membership in the common language. We detail the design of common languages and propose a number of common languages for NP languages. The possibility of nesting common languages is illustrated through a proposal for a co-NP common language. We develop a protocol suite of secure and efficient protocols realizing proofs of membership in the proposed common languages. These protocols can be used when both $X$ and $Y$ have identical or non identical knowledge of the inputs in the common language. The protocols presented are efficient and practical, and can be used for resource constrained networks. This motivates a wide range of applications, including lifetime secrets, and a practical realization of secure two party computation for authentication and others that we propose throughout this thesis.
To my beloved husband Mohamed for his endless support and love, to my kids: Abdelmageed, Faris and Omar for making everything in life sweeter, and to my lovely parents for their unwavering support, love and encouragement over the years. To all of you, thanks for believing in me.
Acknowledgments

I would like to express my deepest appreciation to my advisor, Professor Anish Arora, whose intellect and dedication to quality work will continue to inspire me throughout the years. Thanks for the opportunity to be your student and thanks for always challenging me to be the best that I can. I am greatly indebted to you for your continuous effort to help and your valuable advice in all aspects of life.

I would like to thank my committee members, Professor Dong Xuan, Professor Timothy Carlson and Professor Yinqian Zhang. To Professor Tim I’m really thankful for all your comments and willingness to help, thanks for the 8:00 pm meetings and thorough discussions, it has been greatly influential to my work. And to Professor Dong Xuan, and Professor Yinqian, thanks for all your valuable comments and helpful discussions.

In addition, special thanks to Professor Steve Lai for his help and support and for motivating my interest to work in cryptography and security. I am also greatly thankful to all my friends and colleagues in the Dependable and Distributed Systems lab for all their help, support, valuable discussion and comments on the work. I have learned a lot from each and everyone of them.

To my precious little family, my dear husband and kids. Thanks for all what you’ve endured to help me chase my dreams. It is because of my husband’s support and encouragement, that I was actually able to complete this work. And special thanks to my parents...
whose love and encouragement continue to drive my perseverance and striving for excellence, and to my dad who is my role model and who defines dedication and brilliance in every aspect.
Vita

July 1, 1980 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Born - Cairo, Egypt

2001 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . B.S. Computer Science,
Ain Shams University, Egypt

2006 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . M.S. Computer Science,
Ain Shams University, Egypt

2009-present . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .Graduate Teaching & Research Asso-
ciate, Department of Computer Science &
Engineering, The Ohio State University.

Publications

Research Publications

W. Lin and M. McGrath and I. Ramzy and T. H. Lai and D. Lee, “Detecting Job Interfer-
ence in Large Distributed Multi-Agent Systems – A Formal Approach”. 2013 IFIP/IEEE
International Symposium on Integrated Network Management, IM 2013, pages 411 – 418,
2013.

I. Ramzy and A. Arora, “Using Zero Knowledge to Share a Little Knowledge: Bootstrapping
Trust in Device Networks”. 13th International Symposium, SSS 2011, Grenoble,

and Location Prediction Scheme Exploiting the "One For All" Framework for Location
Management”. 12th International Symposium on Temporal Representation and Reasoning
(TIME), Vermont, USA, 2005.

for Modeling and Updating Location Information in Moving Objects Databases”. Interna-
Fields of Study

Major Field: Computer Science and Engineering

Studies in Cryptography and Network Security: Prof. Anish Arora
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Dedication</td>
<td>iv</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>v</td>
</tr>
<tr>
<td>Vita</td>
<td>vii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xiii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xv</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Proofs on Common Languages: An Alternative Basis of Trust that does not Leak Knowledge Assertions</td>
<td>5</td>
</tr>
<tr>
<td>1.2 Efficient Protocols for Diverse Applications using Proofs on Common Languages</td>
<td>7</td>
</tr>
<tr>
<td>1.3 Lifetime Secrets for Device Networks: No Key Updates?</td>
<td>11</td>
</tr>
<tr>
<td>1.4 Organization</td>
<td>12</td>
</tr>
<tr>
<td>2. Background</td>
<td>13</td>
</tr>
<tr>
<td>2.1 Preliminaries</td>
<td>13</td>
</tr>
<tr>
<td>2.2 Interactive Proofs</td>
<td>15</td>
</tr>
<tr>
<td>2.2.1 NP and IP</td>
<td>16</td>
</tr>
<tr>
<td>2.2.2 Turing Machine Model</td>
<td>18</td>
</tr>
<tr>
<td>2.3 Variants of Interactive Proofs</td>
<td>20</td>
</tr>
<tr>
<td>2.4 Multiprover IP</td>
<td>21</td>
</tr>
<tr>
<td>2.5 Interactive Proofs of Knowledge and Other Variants</td>
<td>22</td>
</tr>
<tr>
<td>2.6 Zero Knowledge</td>
<td>23</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>2.7</td>
<td>Non Interactive Zero Knowledge</td>
</tr>
<tr>
<td>2.8</td>
<td>Secure Multiparty Computations</td>
</tr>
<tr>
<td>2.8.1</td>
<td>The Ideal/Real Model Paradigm</td>
</tr>
<tr>
<td>2.8.2</td>
<td>Authentication Assumptions for SMC</td>
</tr>
<tr>
<td>3.0</td>
<td>Proofs on Common Languages</td>
</tr>
<tr>
<td>3.1</td>
<td>The Common Language Model</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Common Knowledge Assertion $T_{t}(X,Y,I)$</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Prover and Verifier Strategies</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Interactive Proof of $T_{t}$</td>
</tr>
<tr>
<td>3.1.4</td>
<td>Common Language and Relation</td>
</tr>
<tr>
<td>3.1.5</td>
<td>Models of Interaction on Common Languages</td>
</tr>
<tr>
<td>3.2</td>
<td>Proofs of $T_{t}$ via Proofs of Knowledge leak Knowledge</td>
</tr>
<tr>
<td>3.3</td>
<td>Interactive Proofs on Common Languages</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Proofs of Membership in a Common Language</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Zero Knowledge Proofs on Common Languages</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Zero Knowledge Proofs on Common Languages: Efficient Secure Two Party Computation for Authentication</td>
</tr>
<tr>
<td>3.4</td>
<td>Multiprover Proofs of Membership in a Common Language</td>
</tr>
<tr>
<td>3.5</td>
<td>Discussion</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Common Languages based on Common Computational Ability</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Verification of $T_{t}(X,Y,I)$</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Designing Common Languages</td>
</tr>
<tr>
<td>4.0</td>
<td>Protocol Design for Proofs on Common NP Languages</td>
</tr>
<tr>
<td>4.1</td>
<td>Our Approach</td>
</tr>
<tr>
<td>4.2</td>
<td>Common Languages for Discrete Log</td>
</tr>
<tr>
<td>4.2.1</td>
<td>DL Common Language for Identical Prover Strategies</td>
</tr>
<tr>
<td>4.2.2</td>
<td>DL Common Language for non Identical Prover Strategies</td>
</tr>
<tr>
<td>4.3</td>
<td>Common Languages for RSA</td>
</tr>
<tr>
<td>4.3.1</td>
<td>RSA Common Language for Identical Prover Strategies</td>
</tr>
<tr>
<td>4.3.2</td>
<td>RSA Common Language for non Identical Prover Strategies</td>
</tr>
<tr>
<td>4.4</td>
<td>Common Languages for Hamiltonian Graphs</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Hamiltonian Graphs Common Language for Identical Prover Strategies</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Hamiltonian Graphs Common Language for non Identical Prover Strategies</td>
</tr>
<tr>
<td>4.5</td>
<td>Common Languages for Non-Isomorphic Graphs</td>
</tr>
<tr>
<td>4.6</td>
<td>Security Analysis and Adversary Model</td>
</tr>
</tbody>
</table>
ECC-SC TL2 Protocol .................................................. 176
ECC-OK TL2 Protocol .................................................. 177
7.4.2 Security Features of the Realizations ....................... 178
7.4.3 Performance Evaluation ...................................... 178
7.4.4 Features and Merits ............................................ 180

8. Discussions and Conclusions ..................................... 183

8.1 Summary and Conclusions ...................................... 183
8.2 Discussion ......................................................... 186
  8.2.1 A Universal Construction for Common Languages .... 186
  8.2.2 Private Versus Public Coin ................................. 186
  8.2.3 Resettable Zero Knowledge ............................... 188
  8.2.4 A Stronger Notion of Zero Knowledge ................. 189
  8.2.5 Practical Evaluation of Proofs on Common Languages 190

Bibliography ............................................................ 191
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Common Languages Model Postulates.</td>
</tr>
<tr>
<td>3.2</td>
<td>Interactive Proof of Membership in a Common Language.</td>
</tr>
<tr>
<td>3.3</td>
<td>Multiprover Interactive Proof of Membership in a Common Language.</td>
</tr>
<tr>
<td>4.1</td>
<td>NP Language of Discrete Log.</td>
</tr>
<tr>
<td>4.2</td>
<td>Specification of Theorems DL1-$\tau$ and DL1-$T_\tau$, and the Common Language DL1-$\hat{L}<em>{T</em>\tau}(X,Y)$.</td>
</tr>
<tr>
<td>4.3</td>
<td>Specification of Theorems DL2-$\tau$ and DL2-$T_\tau$, and the Common Language DL2-$\hat{L}<em>{T</em>\tau}(X,Y)$.</td>
</tr>
<tr>
<td>4.4</td>
<td>NP Language of RSA.</td>
</tr>
<tr>
<td>4.5</td>
<td>Specification of Theorems RSA2-$\tau$ and RSA2-$T_\tau$, and the Common Language RSA2-$\hat{L}<em>{T</em>\tau}(X,Y)$.</td>
</tr>
<tr>
<td>4.6</td>
<td>Specification of Theorems RSA1-$\tau$ and RSA1-$T_\tau$, and the Common Language RSA1-$\hat{L}<em>{T</em>\tau}(X,Y)$.</td>
</tr>
<tr>
<td>4.7</td>
<td>NP Language of Hamiltonian Graphs.</td>
</tr>
<tr>
<td>4.8</td>
<td>Specification of Theorems HML1-$\tau$ and HML1-$T_\tau$, and the Common Language HML1-$\hat{L}<em>{T</em>\tau}(X,Y)$.</td>
</tr>
<tr>
<td>4.9</td>
<td>Specification of Theorems HML2-$\tau$ and HML2-$T_\tau$, and the Common Language HML2-$\hat{L}<em>{T</em>\tau}(X,Y)$.</td>
</tr>
</tbody>
</table>
4.10 co-NP Language of non-Isomorphic Graphs. .......................... 87

4.11 Specification of Theorems NOI-\(\tau\) and NOI-\(T_{\tau}\), and the Common Language
NOI-\(\hat{L}_{T_{\tau}}(X,Y)\). ......................................................... 88

5.1 CDLL Protocol for Identification based on a DL Language for Identical
Prover Strategies ................................................................. 93

5.2 CLHG Protocol for Identification based on a Common Graph Hamiltonian
Language for Identical Prover Strategies ................................. 101

5.3 Signature Scheme based on a Common DL Language (CLSDL) ...... 110

5.4 ZK-CAKE ................................................................. 115

6.1 A-CDLL Protocol for Identification based on Proofs on Common DL Lan-
guage for non Identical Prover Strategies ............................. 121

6.2 A-CRSAL Protocol for Identification based on Proofs on Common RSA
Language for non Identical Prover Strategies .......................... 129

6.3 A-CLHG Protocol for Identification based on Common Graph Hamiltonian
Language for non Identical Prover Strategies ......................... 139

7.1 Bootstrap Protocol Notation .............................................. 167
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Turing Machine Model [44]</td>
<td>20</td>
</tr>
<tr>
<td>3.1 Ali Baba’s Cave [73]</td>
<td>50</td>
</tr>
<tr>
<td>3.2 Modified Cave</td>
<td>52</td>
</tr>
<tr>
<td>6.1 Grid-based Key Management</td>
<td>148</td>
</tr>
<tr>
<td>7.1 Zero-Knowledge Identification Schema: (ZK[r, n, S])</td>
<td>168</td>
</tr>
<tr>
<td>7.2 Levels of Trust</td>
<td>169</td>
</tr>
<tr>
<td>7.3 TL2 Protocol</td>
<td>172</td>
</tr>
</tbody>
</table>
Chapter 1: Introduction

The concept of interactive proof systems [44] involves a prover, Peggy (P), providing a proof of an assertion to a verifier, Victor (V). The classic example of an assertion to be proved is membership of a given input I in a language L in the complexity class NP [44]; proofs of knowledge of an assertion on L [35, 10] are an important variation. To V, the former reveals only the information that I is in L, whereas the latter reveals only the information that P has knowledge of I, without revealing the relation between I and L. Specifically, Zero Knowledge proof systems are of particular interest. ZK is formalized in terms of a simulation, whereby any interaction with an honest prover, P, can be efficiently simulated without interaction with P.

Proof systems for different languages have been used to develop protocols of identification based on I. In such systems, the prover entity provides a proof of identity, hence establishing identity-based trust with any verifier. Protocols for identification are particularly important for applications where verifiers are assumed to have no knowledge of their own. ID-based cryptography has since gained its popularity from the fact that verifiers need not take any extra steps to verify a prover’s identity, since a proof of identity presented by a prover is all that is needed to perform the verification.
During a typical authentication process, the authentication server (the verifier) can directly or indirectly learn the actual identity of the user (the prover) who is being authenticated. However, the user might not want any one to know his identity but still be able to authenticate himself. I.e., privacy preserving authentication or anonymous authentication may be of interest, particularly in the context of Pervasive Computing systems and networked things.

In today’s world of networked things, assumptions can be made on knowledge of verifiers. The focus becomes on establishing trust within a group or across groups of networked entities. We study and investigate the different aspects and dimensions of the following question:

"Assume Victor is as knowledgeable as Peggy on some common values, can the commonality of the knowledge of Peggy and Victor be used to build more efficient and secure systems?"

In the original model of proof systems, only Peggy possesses relevant knowledge, and traditional proofs of knowledge allow Peggy to assure Victor that it has identifying knowledge on a claimed identity, i.e., an interaction between Peggy and Victor in a proof of knowledge proves the assertion $\tau$: "Peggy has knowledge of a given value $I$". The assertions can be generalized to be on the computational abilities of Peggy rather than on its knowledge. Traditionally, proofs of computational abilities allow Peggy to assure Victor that Peggy has computational ability on a given value or sets of values.

In our model, Victor also has knowledge or computational ability of its own. We consider an assertion on assertions of knowledge, likewise, of computational ability $T_\tau$ as follows: "Both Peggy and Victor can prove knowledge or computational ability for a given value $I$".
In this dissertation, we show that the common knowledge of Peggy and Victor forms a basis of trust that can be used to build systems that are more secure and efficient. The knowledge of Peggy can be viewed as a key that unlocks a secret room, which Victor can also unlock using his own secret key. The room may have one door, in that case, the keys of Peggy and Victor are identical, or may have two doors, for non identical keys at Peggy and Victor. This secret room can be used for Peggy and Victor secret meetings and agreements, hence, it acts as a basis of trust between them.

Our thesis contributions include the following (terms below are defined in Chapter 2 and the relevant chapter in which the term appears):

- We present formal definitions for a new model, the common language model, based on common knowledge of Peggy and Victor. The set of values for which \( T_\tau \) is true defines a new language, which we denote as the common language of Peggy and Victor. The relation of the new language controls the way the knowledge or abilities of Peggy and Victor are related.

- Using the common language model we introduce a new interactive proof system for proving membership in a common language. We generalize the proof system for the multiprover setting, and present a modified definition of zero knowledge for the new proof system based on the common language model.

- We consider cases where Victor and Peggy possess identical knowledge or computational abilities, and also when they have different knowledge or computational abilities on the values of the common language, and demonstrate our approach for deriving a common language relation using possibly nonidentical knowledge. It should
be noted that our approach does not assume any dependency or constraints on the way Peggy and Victor obtain their knowledge or ability.

- Based on these concepts we construct secure identification protocols that ensure privacy, eschew the use of certification from a Key Authentication Center, and allow Peggy and Victor to have independent strategies of their own, which can be used in proofs of membership assertions on common languages. Proofs on common languages thus provide an alternative basis of trust other than identities. Our protocol suite include protocols for symmetric cryptographic systems, and others for asymmetric cryptographic systems.

- We motivate a number of applications for proofs on common languages. Applications include authenticated key exchange (AKE), secret handshakes, delegation based on common languages as well as a secure multiparty computation for authentication.

- We propose and formalize a notion of lifetime secrets as well as motivate the need for these secrets in device networks. We focus on characterizing the necessary and sufficient conditions for lifetime secrets in device networks with a special emphasis on the use of the new model. We present an application for bootstrapping trust in device networks based on life time secrets using a variant of traditional zero knowledge protocols for proofs of knowledge in our common language model but with relaxed assumptions. The proposed application demonstrates that the use of the common language model with relaxed assumptions can improve the security features of traditional zero knowledge protocols of identification.
1.1 Proofs on Common Languages: An Alternative Basis of Trust that does not Leak Knowledge Assertions

Interactive proof systems [44, 35, 10] have been studied extensively over the years. Our work can be viewed as part of the efforts focusing on modeling variants of interactive proofs and arguments.

Proof systems of membership in an NP language \( L \) makes no assertions on the knowledge of the prover, and is based solely on the intrinsic properties of the members of the language. These proof systems guarantees universal verification, i.e., any polynomial time verifier can successfully verify a valid proof regardless of his state of knowledge. On the other hand, proofs of knowledge are intended for a prover \( P \) to prove its identity \( I \) to any verifier \( V \) without shared keys by demonstrating its distinguishing knowledge associated with \( I \), a common input of both \( P \) and \( V \). For these proofs, no assumption is made on the knowledge of \( V \), i.e., \( P \) can be authenticated at any \( V \) regardless of the state of knowledge of \( V \).

In any of the traditional proof systems, a Key Authentication Center (KAC) has to generate the setup parameters for as well as certify the legitimate inputs, the identities \( I \). Without \( KAC \) signing the identities, proofs of knowledge for identification degenerates to \( V \) always accepts, this is because, for any \( P \) there is always \( I \) for which \( P \) has knowledge, eg., proofs of knowledge of discrete log. Hence, certifying the acceptable identities is essential for a sound identification system using proofs of knowledge.

An important generalization of proofs of knowledge, proofs of computational ability [76, 11] allow a prover to convince a verifier that it can satisfy a given relation \( R \) for each input in a certain domain. That is, to prove that it has the “ability” to solve the relation \( R \). The soundness requirement for these proofs is that a knowledge extractor, given any input
from the domain and an accepting view, can extract by interacting with the prover a witness that satisfies $R$ within a bounded expected time.

Instead of authentication based on identity, authentication based on common languages is of interest. A party’s identity is one basis of authentication, whereas we propose mutual knowledge or computational ability of a party with another party on some common values as an alternative basis. The set of common values form a common language whose relation defines the way the knowledge of parties are related on the common values. This forms a basis of mutual authentication that takes the form of an interactive proof. In our work we introduce the notion of proofs on common languages for establishing trust between the prover and the verifier entities for the purpose of identification/authentication.

In our model, Peggy does not trust Victor regardless of Victor’s knowledge state. Nonetheless, both Peggy and Victor have knowledge and trust is established if their independent knowledge can be related on a common input; mutual identity “$I$”. The knowledge of Peggy and Victor can be generated independently, possibly by two different key generation centers, and may or may not be identical. Establishing trust based on common languages does not require a KAC for signing or certifying identities, instead, legitimacy of an $I$ is determined by it being a mutual identity.

Proofs of knowledge or proofs of computational ability can not be used to construct proofs on common languages since they leak assertions of knowledge on the parties involved. For example, assume party $X$ wants to be authenticated at party $Y$ based on knowledge of a given common value $I$ of $X$ and $Y$. If $X$ sent a proof of knowledge to $Y$, then $Y$ not only learns that both $X$ and $Y$ can prove knowledge on $I$, but also the fact that $X$ has knowledge. However, we desire a proof system that does not make explicit knowledge or computational ability assertions, i.e., $Y$ should learn nothing beyond the validity of the
assertion: "X and Y can prove knowledge of I". Preventing leakage of knowledge assertions makes these proof system resilient to a wide range of attacks that proofs of knowledge or ability were prone too, e.g., resetting attacks.

1.2 Efficient Protocols for Diverse Applications using Proofs on Common Languages

General applications of interactive proof systems include secure entity authentication and identification, signature schemes, secret handshakes and others.

**Authentication**: Identification and entity authentication have been studied extensively in the literature, with solutions ranging from the use of passwords for weak authentication, to using challenge-response identification for strong authentication using symmetric or public key techniques, to zero knowledge identification protocols that addressed a number of concerns in the former approaches. Zero knowledge identification protocols are instances of interactive proof systems [44, 35, 10], several such protocols have been proposed over the years [59, 68, 35, 69, 12, 50] in an asymmetric setting.

Our approach to authentication involves a secure two party computation where common language authentication designates the computable function. Zero Knowledge proofs on common languages can be used for anonymous or privacy preserving authentication and has strong security properties. Up to our knowledge, secure two party computations have not been used in this form of authentication previously.

Zero-knowledge proofs are both convincing and yet yield nothing beyond the validity of the assertion being proven. Their applicability in the domain of cryptography is vast; they are typically used to force malicious parties to behave according to a predetermined
protocol. ZK proofs have been developed for a number of identification protocols [59, 68, 35, 69, 12, 50] and received increasing recognition for their security with respect to the use of encrypted passwords or challenge-response authentication schemes. Many variants of ZK proof systems have also been developed for other applications.

Zero Knowledge Proofs on common languages realize a special purpose secure two party computation for set intersection that can be easily extended to define a secure multi-party computation (SMC). Our common language realization is not based on Yao’s garbled circuits technique. The idea is to model authentication based on common languages as a computable function and derive a number of constraints to ensure a secure two party computation of the authentication function. In the literature, SMC was used for anonymous authentication and to prevent server breaches by using multiple authentication servers that compute a joint server authentication function. Our proof system ensures a secure two party computation that achieves anonymous authentication, and is resistant to server breaches, yet does not use multiple authentication servers. Our secure two party computation involves both the authenticator and the party to be authenticated, and the authentication function is based on the common language between both parties. To the best of our knowledge no Secure Multiparty Computation (SMC) has been used in this form before.

Security of two-party and multi-party computation has been studied extensively since it was first introduced by Yao [75] and extended by Goldreich, Micali, and Wigderson [42], and by many others. Nonetheless, most of the work for (SMC) use a combinatorial circuit for presenting the computation problem, and then the parties run a short protocol for every gate in the circuit. The generality and simplicity of this approach has led to its wide spread but has also put its practicality in question as the protocols proposed depend greatly on the size of the circuit, which depends on the size of the input domain, hence, raising concerns
for the complexity of expressing such a computation. Our approach realizes a secure two-party computation using zero knowledge proofs on specific common languages rather than general forms of combinatorial circuits.

For a long time, secure two-party and multi-party computation (SMC) research has mostly been focused on theoretical studies, and few applied problems have been studied. In the literature, there are a few examples of secure multi-party computation problems, such as the Private Information Retrieval problem (PIR) [26, 16, 48], privacy-preserving statistical database [33, 22, 39], and privacy preserving data mining [54, 56, 31].

SMC has also been proposed for authentication to circumvent breach attacks on authentication servers. The user information is distributed among the authentication servers, using a secret sharing scheme, in such a way that none of the authentication servers individually possesses all the information of a user. However, these authentication servers can validate the user using some SMPC arithmetic operations. We demonstrate an alternative use of SMC for authentication that does not require multiple servers and is resilient to breach attacks on the authentication servers using proofs on common languages.

**Authenticated Key Exchange and Secret Handshakes:** ZK schemes have previously been leveraged in the context of password based authenticated key exchange (PAKE), with protocols such as EKE, SPEKE and their variants, yet, none of these AKE protocols uses the ZKP primitives directly, instead, they either use zero knowledge schemes as black boxes added to the original protocol, hence introducing an overhead, or use other constructs that claim the zero knowledge property without sufficient proofs, typically, the Diffie Hellman (DH) constructs. For example, EKE based protocols have the drawback of leaking partial
information about the password to a passive attacker. In contrast, in the SPEKE protocol, an active attacker can test multiple passwords in one protocol execution. Furthermore, neither accommodates short exponents securely or have proofs of security that impose restrictive assumptions [13] or that relax security requirements [19]. One protocol that made use of ZKP, the J-PAKE [47], proposed in 2008 and presented again in 2010 and 2011, uses Zero-Knowledge (ZK) Schnorr’s signature. J-PAKE requires substantial computation and random number generation. Also, it has been shown [71] that it is vulnerable to a password compromise impersonation attack as well as replay and Unknown Key-Share (UKS) attacks. It should be noted that J-PAKE does not use ZKP to derive the session key, but only to prove knowledge of the shared secret/password; if the password is correct, then a shared key is derived based on the password.

We propose a use for proofs on common languages for AKE, where the secure two party computation achieved by our proofs are extended to include a shared key.

On the other hand, affiliation-hiding authentication or secret handshakes [6] introduced a notion of privacy for public key based authentication schemes where the need to exchange credentials as public keys inevitably reveals an entity’s affiliation (for instance, its certification authority). Our common languages proofs can be used to provide efficient and secure realizations for secret handshakes based on the common languages.

We also explore new uses for our proof systems, including delegation, in which $V$ can trust $P$ to act as its representative/delegate upon authentication based on common languages. This has many potential uses in cloud-based systems. Particularly those that employ the hybrid cloud model.
1.3 Lifetime Secrets for Device Networks: No Key Updates?

Authentication is used to establish trust among devices based on cryptographic secrets owned by these devices. Nonetheless, the way in which these secrets are used for bootstrapping trust has a direct consequence on the key management overhead of these systems.

Conventional cryptographic protocols that make direct use of the devices’ original secrets constantly leak information about these secrets. This follows from the computational security assumptions underlying practical cryptography. The problem affects both symmetric and asymmetric secrets but is particularly crucial for symmetric secrets cryptography which is the ubiquitous choice of most device networks. Over time, periodical key updates are necessary to maintain the security of these systems, which incurs significant overheads, especially for device networks—whose numbers, size, and day-to-day influence is growing rapidly.

Solutions that automate and/or reduce key management are therefore important. Ideally, due to the pervasive nature of these systems, one would like to maintain the devices’ secrets for lifetime while being able to use them to establish, restore or maintain trust. This can be achieved if such secrets are used in a way that leaks zero knowledge regardless of how many times these secrets are used, which extends the secrets lifetime to the entity’s lifetime.

We study the feasibility of lifetime secrets in device networks and propose an application for bootstrapping trust based on these lifetime secrets using traditional zero knowledge proof systems. Our solution uses a weakened notion of common languages, where device nodes are allowed to share secrets (following our new model), but assertions of knowledge are explicitly made through traditional zero knowledge proofs of knowledge. Our findings indicate that even if traditional proofs systems are used, a knowledgeable Victor can still make a difference in terms of protocol soundness and efficiency.
1.4 Organization

The rest of the dissertation is organized as follows: In Chapter 2 we present preliminaries and background. In Chapter 3 we introduce the common language model and proposes a proof system based on the new model. In chapter 4 we detail our design approach for protocols realizing proofs on common languages, and demonstrate our approach by presenting a number of common language designs for languages in NP and co-NP. In Chapter 5 we give constructions and proofs of security for efficient new protocols for proofs on common languages in symmetric cryptographic systems, we present protocols for asymmetric cryptographic systems in Chapter 6. In Chapter 7 we propose the notion of lifetime secrets and define an application for bootstrapping trust in device networks based on lifetime secrets that uses the traditional proofs of knowledge in our new common language model with relaxed assumptions. We make our concluding remarks in Chapter 8.
Chapter 2: Background

In this chapter we provide some general notations, definitions and results that will be used throughout the dissertation. Additional definitions and preliminaries appear within the following chapters as appropriate.

2.1 Preliminaries

The standard notion of non-interactive proofs involves a proof of the validity of a mathematical statement; this proof is verified in a deterministic way as follows: the reader (or "verifier") checks the claimed proof of a given statement and is thereby either convinced that the statement is true (if the proof is correct) or is unconvinced (if the proof is flawed - note that the statement may possibly still be true in this case, it just means there was something wrong with the proof). A statement is true in this traditional setting if and only if there exists a valid proof that convinces a legitimate verifier.

This notion is abstracted as follows: a prover $P$ is trying to convince the verifier $V$ of the validity of some particular statement $I$ (e.g., $I \in L$ for some fixed language $L$). $V$ is expected to run in time polynomial in $|I|$, since we are concerned only with proofs that are efficiently verifiable. A traditional mathematical proof can be cast in this framework by simply having $P$ send a proof $\pi$ to $V$, who then deterministically checks whether $\pi$ is a valid proof of $x$ and outputs $V(I, \pi)$ (with 1 denoting acceptance and 0 rejection). Note that since $V$ runs in
polynomial time, we may assume that the length of the proof $\pi$ is also polynomial. Now, the traditional mathematical notion of a proof can be specified as follows:

- If $I \in L$, then there exists a proof $\pi$ such that $V(I, \pi) = 1$.
- If $I \notin L$, then for any proof $\pi$ the prover sends, $V(I, \pi) = 0$.

We refer to the system $(P, V)$ satisfying the above as a proof system for $L$. $L$ has a proof system as defined above if and only if $L \in NP$ where NP is the class of languages each having a proof system that supports efficient deterministic verification. Interesting proof systems involve the cases when the set of proofs is not empty, hence the verifier sometimes accepts and other times it rejects. When a non empty proof set is combined with a probabilistic verifier strategy (introducing the possibility of error), a rather more interesting class of proofs emerge, namely interactive proofs, which are central to this thesis.

An interaction is a single execution of the proof system, and a strategy for a prover or verifier is a function mapping its view of the interaction so far to a description of its next move; that is, such a strategy prescribes its next move (i.e., its next message or its final decision) as a function of the common input, its internal coin tosses, and all messages it has received so far. Note that this formulation implicitly presumes that each party records the outcomes of its past coin tosses as well as all the messages it has received, and determines its moves based on these [40].
2.2 Interactive Proofs

Interactive proofs (IP) (introduced by Goldwasser, Micali and Rackoff) [44] emerge when the notion of efficient verification, which underlies the notion of a proof system is associated with probabilistic and interactive polynomial-time computations. This association is quite natural in light of the growing acceptability of randomized and distributed computations. Thus, a proof in this context is not a deterministic one, but rather a randomized one in which the verifier interacts with the prover. Intuitively, one may think of this interaction as consisting of questions asked by the verifier, to which the prover has to reply convincingly. The above discussion, as well as the following definition, makes explicit reference to a prover, whereas a prover is only implicit in the traditional definitions of proof systems (e.g., NP-proofs).

An interactive proof can be seen as a game between a probabilistic computationally bounded (polynomial-time) verifier and a computationally unbounded prover that tries to convince the verifier of the validity of some assertion. It is required that if the assertion holds then the verifier always accepts (i.e., when interacting with an appropriate prover strategy). Conversely, if the assertion is false then the verifier must reject with non-negligible/noticeable probability, no matter what strategy is being used by the prover. The soundness error probability can be reduced by (either sequential or parallel) repetitions.

Definition 2.1 (Interactive proof systems for class IP [44]) An interactive proof system for a language $L \in IP$ is a two-party interaction, between a verifier executing a probabilistic polynomial-time strategy (denoted $V$) and a prover which executes a computationally unbounded strategy (denoted $P$), satisfying:
Completeness: For every \( I \in L \) the verifier \( V \) always accepts after interacting with the prover \( P \) on common input \( I \).

Soundness: For some polynomial \( p \), it holds that for every \( I \notin L \) and every potential strategy \( P^* \), the verifier \( V \) rejects with probability at least \( 1/p(|I|) \), after interacting with \( P^* \) on common input \( I \).

The soundness property guarantees that the verification procedure cannot be tricked into accepting false statements, i.e., the verifier can protect itself from being convinced of false statements regardless of what the prover does to fool it. On the other hand, completeness captures the ability of some prover to convince the verifier of true statements (belonging to some predetermined set of true statements). Both properties are essential to the notion of a proof system.

Interactive proof systems are usually referred to as interactive proofs of membership in an NP language, as well as interactive proofs of membership assertions on a given language.

### 2.2.1 NP and IP

Every language in NP has an interactive proof system [14, 34]. Specifically, let \( L \in NP \) and let \( R_L \) be a witness relation associated with the language \( L \) (i.e., \( R_L \) is recognizable in polynomial time and \( L \) equals the set \( \{ I : \exists w \text{ s.t.} |w| = poly(I) \land (I, w) \in R_L \} \)). Then, an interactive proof for the language \( L \) consists of a prover \( P \) that on common input \( I \in L \) sends a witness \( w \), and a verifier that upon receiving \( w \) outputs 1 if \( |w| = poly(|I|) \) and \((I, w) \in R_L \), and 0 otherwise. Clearly, when interacting with this prover, the verifier will always accept inputs in the language. On the other hand, no matter what a cheating prover does, the verifier will never accept inputs not in the language. In NP proof systems both parties are deterministic, i.e., they do not randomize their computations. It is easy to see
that only languages in NP have interactive proof systems in which both parties are deterministic. In other words, NP can be viewed as a class of interactive proof systems in which the interaction is unidirectional (i.e., from the prover to the verifier) and the verifier is deterministic (and never errs). In general interactive proofs, both restrictions are waived: the interaction is bidirectional and the verifier is probabilistic (and may err with some small probability). Both bidirectional interaction and randomization seem essential to the power of interactive proof systems.

The class of problems having interactive proof systems is denoted IP. Note that by repeating such a proof system for \(O(p(|I|^2))\) times, we may decrease the probability that \(V\) accepts a false statement (from \(1 - (1/p(|I|)))\) to \(2 - p(|I|)\). Thus, when constructing interactive proofs we sometimes focus on obtaining a non negligible rejection probability for the “no” instances (i.e., obtaining a soundness error bounded away from 1), whereas when using interactive proofs we typically assume that their soundness error is negligible.

The basic protocol realizing an interactive proof system is a challenge-and-response protocol, consisting of a specified number of rounds. During each round, two parties, the prover and the verifier, alternatively do the following:

- Receive a message from the other party
- Perform a private computation
- Send a message to the other party

A typical protocol round consists of a challenge sent by the verifier, and a response sent by the prover. At the end of the proof, the verifier either accepts or rejects, depending on whether or not the Prover successfully replies to the Verifier’s challenges.
2.2.2 Turing Machine Model

An Interactive Turing Machine (ITM) [44] is a six-tape deterministic Turing machine $M$ with a read-only input tape $I.M$, a read-only random tape $R.M$, a read/write work tape $W.M$, a read-only communication tape $C_r.M$, a write-only communication tape $C_w.M$, and a write only output tape $O.M$. The string that appears on the input tape is called the input. The (infinite) contents of the random tape can be thought of as the outcomes of an infinite sequence of unbiased coin tosses. The string that appears on the output tape when the machine halts is called the output. The contents of the write-only communication tape can be thought of as messages sent by the machine; while the contents of the read-only communication tape can be thought of as messages received by the machine.

The complexity of an interactive Turing machine is measured in terms of its input (i.e., contents of input tape). We will consider the number of steps taken by a Turing machine as a function of its input length, bound this function from above by a "smoother" function, and consider whether or not the bound is (or can be) a polynomial. This puts more emphasis on the typical behavior of the machine rather than exceptionally good behaviors on special short inputs. Hence, an interactive Turing machine $M$ is considered polynomial-time if there exists a polynomial $Q$ such that the number of steps $M$ performs on input $I$ is at most $Q(|I|)$, no matter what the contents of its random tape and read-only communication tape are. When discussing the expected number of steps of an ITM, the expectation is taken over the contents of the random tape only.

A probabilistic Turing machine is an "extended" Turing machine that uses its local configuration to randomly choose its next move (with uniform probability distribution) among a finite number of possibilities. On the other hand, a deterministic Turing machine determines its next move using its local configuration only. A probabilistic TM can be viewed as
a machine that is tossing an unbiased coin before each move and determining the next move using the outcome of the coin. On input $I$, the output of a probabilistic Turing machine $M$ is a random variable defined over the probability space of all possible internal coin tosses. Equivalently, probabilistic Turing machines can be viewed as deterministic machines with two inputs: the ordinary input, and an auxiliary "random input". We consider the probability distributions defined by fixing the first input and letting the auxiliary input assume all possible values with equal probabilities.

Two interactive machines are said to be linked if they have different identities, their input tapes coincide and the read only communication tape of one machine coincides with the write only communication tape of the other machine, and vice versa. The other tapes of both machines (i.e., the random tape, the work tape, and the output tape) are distinct. The joint computation of a linked pair of ITMs, on a common input $I$, is a sequence of pairs. Each pair consists of the local configuration of each of the machines. In each such pair of local configurations, one machine (not necessarily the same one) is active while the other machine is idle.

The interactive Turing machine model shown in Figure 2.1 is used to model the interactions between two parties $P$ and $V$ involved in an interactive proof system [?]. Both $P$ and $V$ are modeled as ITMs. Both have a common input tape, whose value is $I$ and whose length is denoted by $|I|$. $P$ has a a private infinite random tape $R.P$, a private work tape $W.P$, and an append-only communication tape from $P$ to $V$, namely $C.P$. Likewise, $V$ has a private infinite random tape $R.V$, a private work tape $W.V$, and an append-only communication tape from $V$ to $P$, namely $C.V$. Note that $C.P$ and $C.V$ are also read-only communication tapes for $V$ and $P$ respectively.
2.3 Variants of Interactive Proofs

There are a number of variants to interactive proofs (IP). Arthur-Merlin proof systems (or public-coin proof systems, introduced by Babai [66, 5], are a special case of interactive proofs in which the verifier must send the outcome of any coin it tosses in the clear, and no further information is sent. Yet, as shown in [45], this restricted case has essentially the same power as the general case introduced by Goldwasser, Micali and Rackoff. Thus, in the context of interactive proof systems, asking random questions is as powerful as asking tricky questions, although as will be noted later this does not necessarily hold in the context of zero-knowledge proofs. Also, in some sources interactive proofs are defined so that two-sided error probability is allowed (rather than requiring perfect completeness as specified in the above definition); yet, this does not increase their power [45].

![Turing Machine Model](image_url)

Figure 2.1: Turing Machine Model [44]
Arguments (or Computational Soundness) is another fundamental variant on the notion of interactive proofs. It was first introduced by Brassard, Chaum and Crepeau [20], who relaxed the soundness condition in interactive proofs so as to only refer to feasible ways of trying to fool the verifier (rather than to all possible ways). Specifically, the soundness condition was replaced by the following computational soundness condition that asserts that it is infeasible to fool the verifier into accepting false statements. That is:

**Definition 2.2 (Arguments [20])**  
*For every polynomial $p$, every prover strategy that is implementable by a family of polynomial-size circuits $\{C_n\}$, and every sufficiently large $I \in \{0,1\}^* S$, the probability that $V$ accepts $I$ when interacting with $C|I|$ is less than $1/p(|I|)$.*

Although the computational-soundness error can always be reduced by sequential repetitions, it is not true that this error can always be reduced by parallel repetitions. Protocols that satisfy the computational-soundness condition are called arguments. Argument systems may be more efficient than interactive proofs ([20]).

### 2.4 Multiprover IP

The generalized interactive-proof model consists of $k$ computationally unbounded and untrusted provers rather than one, who jointly agree on a strategy to convince the verifier of the truth of an assertion and then engage in a polynomial number of message exchanges with the verifier in their attempt to do so. To believe the validity of the assertion, the verifier must make sure that the $k$ provers can not communicate with each other during the course
of the proof process. Thus, the complexity assumptions made in previous work, have been traded for a physical separation between the $k$ provers [15].

It has been shown that MIP=NEXP, where MIP is the class of languages who have a multi-prover interactive proof and NEXP is the class of languages that can be decided in nondeterministic exponential time.

2.5 Interactive Proofs of Knowledge and Other Variants

Another class of interactive proofs is the “Interactive Proof of Knowledge”, which allow the prover to convince the verifier that it has knowledge related to a certain assertion. This notion is especially useful in languages where the problem of deciding membership is trivial, i.e., when every $I$ is in language. In this case, proving membership in a language does not capture the claim that the prover wants to demonstrate, since even though the assertion is true, anyone can give an acceptable proof. Proofs of knowledge enable the prover to demonstrate knowledge of why an assertion is true.

An interactive proof of knowledge [35], $(P,V)$, seeks to perform a language recognition task [44] where the subset of “good” inputs are not solely defined by an intrinsic property of the proof input $I$ capturing the language $L$ (as is the case for proof systems of language membership), but rather on the state of knowledge of $P$ relative to $I$. Recall that $L$ is the language of inputs $I$ whose witness set $s(I)$ is the set of $s$ such that $(I,s)$ is in the language relation $R_L$.

Proofs of knowledge use the same Turing Machine Model described earlier but add an additional knowledge tape $K.M$ for the Turing Machine $M$. Proofs are accepted on $I$ iff a witness or proof $s$ is “known” to $P$. That is, $P$ tries to convince $V$ that its K.P contains $s$ by offering a proof of $(I,s) \in R_L$, such that for all $P$, for all $I$, and for all $s$ such that $(I,s) \in R_L$: 
• If $P$ knows $s$, $V$ should accept $P$'s proof for $I$ with overwhelming probability.

• If $P$ does not know $s$, $V$ should accept $P$'s proof for $I$ with negligible probability.

**Definition 2.3 (Negligible function)** A function $\varepsilon \rightarrow [0, 1]$ is called negligible if and only if $\forall c \in N \exists n_0 \in N$ such that $\forall n \geq n_0$, $\varepsilon(n) < n^{-c}$.

**Definition 2.4** [35, 10] A pair of polynomial time probabilistic Turing machines $(P, V)$ is called an “interactive proof of knowledge system for relation $R$” if there exists a negligible function $\varepsilon(.) : N \rightarrow [0, 1]$ such that:

- **Completeness:** $\forall (I, s) \in R$, $\text{pr}( (V, P(s))(I) = 1 ) > 1 - \varepsilon(|I|)$

- **Soundness:** $\exists E \forall P \forall I \forall s' : \text{pr}( (V, P(s'))(I) = 1 ) < \text{pr}(E(I, \langle P(s') \rangle) \in s(I)) + \varepsilon(|I|)$.

The probability is taken over the coin tosses of $V$, $P$ and $M$. The knowledge extractor $E$ runs in expected polynomial time, and uses $P$ as a blackbox.

### 2.6 Zero Knowledge

A fundamental measure proposed by Goldwasser et al. [44] is that of the amount of knowledge released by an interactive proof on a given input. Informally, a proof system is called zero-knowledge if whatever the verifier could generate in probabilistic polynomial-time after seeing a proof of membership, it could also generate in probabilistic polynomial-time when merely told by a trusted oracle that the input is indeed in the language. Zero-knowledge proofs have the remarkable property of being both convincing and yielding nothing except that the assertion is indeed valid. Besides being a very intriguing notion, zero-knowledge proofs promise to be a very powerful tool for the design of secure cryptographic protocol. Typically these proofs must cope with the problem of distrustful parties...
convincing each other that the messages they are sending are indeed computed according
to their predetermined local program. Such proofs should be carried out without yielding
any secret knowledge. In particular cases, zero-knowledge proofs have been used to design
secure protocols [44, 28, 59, 68, 35, 69, 12, 50] and have been receiving increasing recogni-
tion for their security with respect to the use of encrypted passwords or challenge-response
authentication schemes.

When considering zero knowledge proofs, the standard notion of non-interactive proofs
does not suffice, because non interactive zero-knowledge proofs exist only for languages
that are easy to decide. ZK proofs are proofs that yield nothing beyond the validity of the
assertion. That is, a verifier obtaining such a proof only gains conviction in the validity of
the assertion. This is formulated by saying that anything that is feasibly computable from
a zero-knowledge proof is also feasibly computable from the (valid) assertion itself, by a
so-called simulator. Variants on the basic definition includes:

- consideration of auxiliary inputs;
- mandating of universal and black-box simulations;
- restricting attention to honest (or rather semi-honest) verifiers;
- the level of similarity required of the simulation.

Towards formally defining ZK, we must first define what the verifier sees, or its view
of an interaction with an honest prover while participating in the proof. A view includes
the contents of all the read only tapes of V, i.e., the input tape, random tape and read only
communication tape. Since randomization is involved, a verifier will not necessarily get the
same view each time it takes part in the proof interaction, so we focus on the probability
distribution of the views rather than the values themselves. I.e., the probability ensemble
associated with the set of views.

**Definition 2.5 (Ensemble X [43])**

An ensemble $X = \{X_n\}_{n \in \mathbb{N}}$ is a sequence of random
variables each ranging over binary strings. (we sometimes choose to omit $n \in \mathbb{N}$ from the
notation)

A dishonest verifier can cheat, and we’ll need to generate views that have distribution
that is similar to the distribution for all possible verifiers $V$, i.e., $\text{VIEW}_V$. In then follows
that the overall views distribution will be affected, since a verifier, particularly a dishonest
one, may not chose the values randomly when given the option. The verifier could base
each decision off the previous or fix it to a value.

A proof system is said to be zero knowledge if every possible accepting verifier view
$\text{VIEW}_V$ can be generated in polynomial time with an indistinguishable distribution. An
expected polynomial time machine that can generate $\text{VIEW}_V$ is called a simulator. If we
let $\{(P, V)(I)\}_{I \in L}$ be the probability distribution ensemble on input $I$, random and com-
municate tapes of any Verifier given a truthful prover of $I \in L$, then it is equivalent to the
distribution of $\text{VIEW}_V$.

Zero knowledge uses a simulator machine, $M$, for any verifier, $M_V$, that will output val-
ues of a close-to-equivalent distribution, $\{M_V(I)\}_{I \in L}$. If $\{M_V(I)\}_{I \in L}$ and $\{(P, V)(I)\}_{I \in L}$
are at least polynomially indistinguishable then the proof system is proven zero-knowledge.
The strength of zero-knowledge depends on how close the distributions are.
Informally speaking, the simulation paradigm (introduced in [44]) states that a protocol is secure if the view of any adversary in a real protocol execution can be generated solely from the information the adversary legitimately possesses (i.e., its input and output). This is demonstrated by presenting a simulator that is given only the input and output of the adversarial (or corrupted) party or parties, and generates a view that is indistinguishable from the view of the adversary in a real protocol execution. The implication is that the adversary learns nothing from the protocol execution, since it could anyway generate everything that it sees in such an execution by itself.

Definition 2.6 [(Computational indistinguishability [41, 43])]

Let \( s \subseteq \{0, 1\}^* \) be an infinite set of strings, and let \( \pi_1 = \{\pi_1(I)\}_{I \in s} \) and \( \pi_2 = \{\pi_2(I)\}_{I \in s} \) be two probability ensembles. For every algorithm (test) \( D \), let \( P_D^1 \) denote the probability that \( D \) outputs 1 on input \( I \) and an element chosen according to the probability distribution \( \pi_i(I) \).

\[
P_D^1(X) = \sum_{\alpha} \Pr(D(I, \alpha) = 1) \Pr(\pi_i(I) = \alpha)
\]

The ensembles \( \pi_1 \) and \( \pi_2 \) are polynomially indistinguishable if for every expected polynomial-time algorithm \( D \), for every constant \( c > 0 \), and for all sufficiently long \( I \in s \).

\[
|P_D^1(X) - P_D^2(X)| \leq |I|^{-c}
\]

Definition 2.7 (Computational Zero Knowledge [41])

Let \( (\mathcal{P}, V) \) be an interactive proof system for a language \( L \), and \( V \) be an arbitrary TM. Denote by \( \{(\mathcal{P}, V)(I)\} \) the probability ensemble on all the read-only tapes (\( \text{VIEW}_V \)) of \( V \), when interacting with honest \( \mathcal{P} \) on common input \( I \in L \). We say that the proof system \( (\mathcal{P}, V) \) is computational zero-knowledge (for \( L \)) if for all expected polynomial-time TM \( V \), there exists an expected polynomial-time machine \( M_V \), such that the probability ensembles \( \{M_V(I)\}_{I \in L} \) and \( \{(\mathcal{P}, V)(I)\}_{I \in L} \) are
Definition 2.8 (Auxiliary Input Computational Zero Knowledge [41]) Let \((\overline{P}, V)\) be an interactive proof system for a language \(L\), and \(V\) be an arbitrary TM. Denote by \(\{(\overline{P}, V(\hat{z})(I))\}\) the probability ensemble on all the read-only tapes \((\text{VIEW}_V)\) of \(V\) and auxiliary input \(\hat{z}\), when interacting with honest \(\overline{P}\) on common input \(I \in L\) and for all \(\hat{z} \in \{0,1\}^*\).

We say that the proof system \((\overline{P}, V)\) is computational zero-knowledge (for \(L\)) if for all expected polynomial-time TM \(V\), there exists an expected polynomial-time machine \(M_V\) such that the probability ensembles \(\{M_V(I, \hat{Z})\}_{I \in L}\) and \(\{(\overline{P}, V)(I)\}_{I \in L}\) are polynomially indistinguishable.

Definition 2.9 (Perfect Zero Knowledge [41]) Let \((\overline{P}, V)\) be an interactive proof system for a language \(L\), and \(V\) be an arbitrary TM. Denote by \((\overline{P}, V)(I)\) the probability ensemble on all the read-only tapes \((\text{VIEW}_V)\) of \(V\), when interacting with honest \(\overline{P}\) on common input \(I \in L\). We say that the proof system \((\overline{P}, V)\) is perfect zero-knowledge (for \(L\)) if for all expected polynomial-time TM \(V\), there exists an expected polynomial-time machine \(M_V\) such that the probability ensembles \(\{M_V(I)\}_{I \in L}\) and \(\{(\overline{P}, V)(I)\}_{I \in L}\) are equivalent.

Definition 2.10 (Statistical Zero Knowledge [41]) Let \((\overline{P}, V)\) be an interactive proof system for a language \(L\), and \(V\) be an arbitrary TM. Denote by \((\overline{P}, V)(I)\) the probability ensemble on all the read-only tapes \((\text{VIEW}_V)\) of \(V\), when interacting with honest \(\overline{P}\) on common input \(I \in L\). We say that the proof system \((\overline{P}, V)\) is statistical zero-knowledge (for \(L\)) if for all expected polynomial-time TM \(V\), there exists an expected polynomial-time machine \(M_V\) such that the probability ensembles \(\{M_V(I)\}_{I \in L}\) and \(\{(\overline{P}, V)(I)\}_{I \in L}\) are statistically indistinguishable.
close. I.e., the statistical difference between them is less than $|I|^c$ for all $c > 0$ and sufficiently large $I$.

Zero knowledge (ZK) proof systems have been built using bit commitment schemes, which allows deriving zero knowledge proof systems for all languages in NP. Bit commitment schemes involve committing to a secret bit, and can be implemented either by using any one-way function existing or by using other means (possibly physical) for hiding information. This result demonstrates the generality of zero-knowledge proofs, and is the key for their wide applicability to cryptography and related fields. The strength of zero-knowledge proof systems is that they exist "independently of cryptography and number theory", i.e., for example, there are perfect ZK proof systems for the problems of graph isomorphism and graph nonisomorphism that uses no computational assumptions [41].

**Definition 2.11 (A non-uniform Probabilistic Polynomial Time Turing Machine PPT)** is allowed to specify different machines for different input lengths, as opposed a single machine that must work for any input length. I.e., it is a sequence of probabilistic Turing Machines $\{M_1, M_2, \ldots\}$ such that $\exists c \in N$ such that $\forall x, M_{|x|}(x)$ halts in $(|x|)^c$ steps.

**Definition 2.12 (One-Way Function)** A function $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ is one-way if and only if there exists polynomial-time $M$ such that $\forall x \ M(x) = f(x)$, assuming $x$ is drawn uniformly at random from the set $\{0, 1\}^n$, and for all non-uniform PPT $A$, it follows that: $\Pr[A(f(x), 1^n) \in f^{-1}(f(x))] = \varepsilon(|x|) = \varepsilon(n)$
On the other hand, Brassard and Crepeu used the intractability assumption of quadratic residuosity to derive zero-knowledge proof systems for all languages in NP [21]. Brassard et al showed that, if factoring is intractable, then every NP language has a perfect zero-knowledge argument. The difference between an argument and interactive proof systems is that in an argument the soundness condition is restricted to a probabilistic polynomial-time machines with auxiliary input which is fixed before the protocol starts. Hence it is computationally infeasible to fool the verifier into accepting with nonnegligible probability an input not in the language. Brassard et al. also proposed an interesting application of perfect zero knowledge arguments in settings in which the verifier may have infinite computing power while the prover is restricted to polynomial-time computations.

There has been efforts focusing on modeling variants of zero knowledge proofs and arguments. One variant is related to relaxing the notion of zero knowledge as in [37, 32, 17, 53, 8, 60]. Witness indistinguishability, first introduced by Feige and Shamir [37], refers to proofs where the view of the verifier is oblivious to the witness the honest prover uses. In [8, 60], the notion of quasi-polynomial time simulatability is the basis of their weakened notion of ZK. Quasi-polynomial time simulatable proofs only “leak” information that can be calculated in quasi-polynomial time. On the other hand, in [32, 17, 53], the notion of “Weak ZK” mostly focuses on a simulation model that uses the verifier to determine the challenge and simulate accordingly.

The recent efforts on leakage-resilient identification and authenticated key agreement protocols in [38, 1, 29, 30] provide interesting characterization of leaked knowledge for ZK proofs.
2.7 Non Interactive Zero Knowledge

Non-Interactive Zero Knowledge (NIZK), introduced by Blum, Feldman, and Micali in 1988, is a fundamental cryptographic primitive which has attracted considerable attention in the last decade and has been used throughout modern cryptography in several essential ways. NIZK plays a central role in building provably secure public-key cryptosystems based on general complexity-theoretic assumptions that achieve security against chosen ciphertext attacks. In a multi-party setting, given a fixed common random string of polynomial size which is visible to all parties, NIZK allows an arbitrary polynomial number of Provers to send messages to polynomially many Verifiers, where each message constitutes an NIZK proof for an arbitrary polynomial-size NP statement. The model, introduced in [18], consists of three entities: a prover, a verifier and a uniformly selected sequence of bits (selected by a trusted third party). Both verifier and prover can read the random sequence, and each can toss additional coins. The interaction consists of a single message sent from the prover to the verifier, who then is left with the decision (whether to accept or not). Based on some reasonable complexity assumptions, one may construct non-interactive zero-knowledge proof systems for every NP-set ([18, 36]).

The Fiat-Shamir (FS) transform is a popular tool to produce particularly efficient digital signature schemes out of identification protocols. It is known that the resulting signature scheme is secure (in the random oracle model) if and only if the identification protocol is secure against passive impersonators. A similar results holds for constructing ID-based signature schemes out of ID-based identification protocols.
Several types of security requirements were considered in the literature for signature schemes. We say that a signature scheme is secure if it is existentially secure against adaptive chosen message attacks.

**Definition 2.13 (Security against adaptive chosen message attacks)** A signature scheme \(\text{SIG} = (\text{GEN}, \text{SIGN}, \text{VERIFY})\) is secure if for every polynomial-size circuit family \(F = \{F_n\}, n \in N,\) with oracle access to \(\text{SIGN},\) the probability that, on input a uniformly chosen verification-key \(V K \leftarrow \text{GEN}(1^n),\) \(F_n\) outputs a pair \((M_0, \text{SIG}_{M_0})\) such that \(\text{VERIFY}(V K, M_0, \text{SIG}_{M_0}) = 1\) and such that \(M_0\) was not sent by \(F_n\) as an oracle query to \(\text{SIGN},\) is negligible (where the probability is over \(V K\) and over the randomness of the oracle \(\text{SIGN}).\)

### 2.8 Secure Multiparty Computations

The growth of the Internet has triggered tremendous opportunities for cooperative computation, where people are jointly conducting computation tasks based on the private inputs they each supplies. These computations could occur between mutually untrusted parties, or even between competitors, e.g., customers might send to a remote database queries that contain private information; two competing financial organizations might jointly invest in a project that must satisfy both organizations’ private and valuable constraints, and so on. Today, to conduct such computations, one entity must usually know the inputs from all the participants; however if nobody can be trusted enough to know all the inputs, privacy will become a primary concern.
For these systems a number of distinct distrusting connected computing devices (or parties) wish to carry out a joint computation of some function of interest. A secure multiparty computation enables the involved parties to jointly perform the computation while preserving the privacy of their individual inputs. In the setting of secure two-party computation, two mutually distrusting parties wish to compute some function of their inputs while preserving, to the extent possible, various security properties such as privacy, correctness, and more. Secure two-party computation is a special case for Secure Multiparty Computation (SMC) which provides a powerful and general tool for distributing computational tasks between mutually distrusting parties without compromising the privacy of their inputs. Moreover, zero-knowledge proofs can be viewed as a special case of secure two-party computation, where the computable function involves verifying the validity of the prover’s knowledge on a given input $I$ [49].

The aim of a secure multiparty computation task is for the participating parties to securely compute some function of their distributed and private inputs. Security is mainly defined in terms of privacy of the inputs, correctness of the output, though its possible to include different properties, as fairness, independence of the input and others. These security properties must be guaranteed even when some subset of the parties and/or an external adversary maliciously and actively attack the protocol with the aim of compromising the uncorrupted parties’ security. Instead of having to define the security properties per application, which often results in non complete lists of security properties. SMC uses the real world and ideal world paradigm as an alternative to modeling security.

This research area has produced a rich body of work that deals with many aspects of the problem, including definitional issues, general feasibility results, protocols with low round
and communication complexity, protocols that rely on a wide variety of computational assumptions, lower bounds, and security under composition.

### 2.8.1 The Ideal/Real Model Paradigm

This has been the dominant paradigm in the investigation of secure computation [23]. Informally, this paradigm defines the security of a real protocol by comparing it to an ideal computing scenario in which the parties interact with an external trusted and incorruptible party. In this ideal execution, all the parties send their inputs to the trusted party (via ideally secure communication lines). The trusted party then computes the function on these inputs and sends each party its specified output. Such a computation embodies the goal of secure computation.

Specifically, it is clear that the privacy property holds, because the only message that a party receives (even one who behaves maliciously) is the output. Likewise, since the trusted party is incorruptible, correctness is also guaranteed to hold. In addition to the fact that the above and other properties are preserved in an ideal execution, the simplicity of the ideal model provides an intuitively convincing security guarantee. Note that the only message that a party sends in an ideal execution is its input, and so the only power that a corrupted party has is to choose its input.

However, in the real world, there is no external party that can be trusted by all parties. Rather, the parties run some protocol amongst themselves without any outside help. We consider a real protocol that “emulates” an ideal execution to be secure. That is, we say that a real protocol that is run by the parties (in a world where no trusted party exists) is secure, if for any adversary carrying out a successful attack on a real protocol, there exists an adversary that successfully carries out the same attack in the ideal world. This suffices
because, as we have seen, no successful attacks can be carried out in an ideal execution. Thus, no successful attacks can be carried out on the real protocol, implying that it is secure.

2.8.2 Authentication Assumptions for SMC

A common approach for most of the SMC protocols is to treat the authenticated communication aspect of the problem as extraneous to the actual protocol design. That is, practically all the existing protocols assume that the parties can communicate in an ideally authenticated way: the adversary is assumed to be unable to send messages in the name of uncorrupted parties, or modify messages that the uncorrupted parties send to each other. This means that authentication must be provided by some mechanism that is external to the protocol itself. In [7], the authors propose a general methodology for incorporating different authentication mechanisms in protocols. This methodology is based on incorporating the authentication within the protocol itself, rather than confining it to a separate module.
In this chapter we introduce the common language model and define a new interactive proof system for proving membership in a common language. We also propose a generalization for our proof system in the multiprover setting. Based on the characteristics of the new proof system, we redefine the notion of zero knowledge for interactive proofs. The definition provided is for a computational zero knowledge proof of membership in a common language, it can be readily extended to different variants of zero knowledge.

3.1 The Common Language Model

Informally speaking, a common language \( L \) of parties \( X \) and \( Y \) is the set of values for which the knowledge of \( X \) and \( Y \) can be related, and \( R_L \) is the corresponding relation of the common language. In this section we detail our common language model in terms of the common knowledge assertion, strategies for proving and verifying the common knowledge assertion, as well as the common language and the underlying relation.

3.1.1 Common Knowledge Assertion \( T_\tau(X,Y,I) \)

Let \( \tau(M,I) \) on party \( M \) and input \( I \in L \) be the assertion that \( M \) has problem specific knowledge of \( I \). We assume that the validity of \( \tau(M,I) \) can be checked in polynomial time.

\(^1\)As we shall see later, \( L \) is in essence a refinement of the language associated with some computationally hard problem at hand.
time. The common knowledge assertion $T_\tau(X,Y,I)$ is then defined as follows:

**Definition 3.1 (Common Knowledge Assertion $T_\tau(X,Y,I)$):** For parties $X$, $Y$ and input $I$, $T_\tau(X,Y,I)$ is the assertion that $\tau(X,I)$ is true and that $\tau(Y,I)$ is true.

It then follows that if $T_\tau(X,Y,I)$ is true, then the application specific knowledge of each of $X$ and $Y$ are related through $I$. The following notation will be used in our formulations.

- $\bar{X}$: An honest party that has a polynomial-time program, follows its protocol, has an efficient prover strategy for proving $\tau(\bar{X},I)$ for some value $I$.
- $\bar{Y}$: An honest party that has a polynomial-time program, follows its protocol, has an efficient prover strategy for proving $\tau(\bar{Y},I)$ for some value $I$.
- $\tilde{X}$: A dishonest party has an arbitrary polynomial-time program and does not follow its protocol.
- $\tilde{Y}$: A dishonest party has an arbitrary polynomial-time program and does not follow its protocol.
- $X$: Either $\bar{X}$ or $\tilde{X}$.
- $Y$: Either $\bar{Y}$ or $\tilde{Y}$.

### 3.1.2 Prover and Verifier Strategies

We shall use $(X, Y)(I)$ to denote ordered two party interaction between $X$ and $Y$ on input vector $I$. The parties $X$ and $Y$ are modeled as Turing Machines (TM) following the original TM model outlined in Chapter 2. We assume that TM $Y$ starts the computation.
and both $X$ and $Y$ take turns in being active. When a TM is active, it performs some private calculations, and may or may not send out a message to the other party. Hence, the TMs run their programs and write on their tapes including their communication tapes if necessary. As in the traditional model, the interaction can be terminated by either $X$ or $Y$ at any time, and the probability of each calculation in the interaction is taken over the coin tosses of both $X$ and $Y$.

Unlike the traditional model, we do not designate any of the parties as a prover or a verifier. We abstract the proving and verifying abilities and identify them as party’s strategies. Hence, it is possible for both $X$ and $Y$ to be both a prover and a verifier if each of them have the necessary efficient prover and verifier strategies. Below we define the prover and verifier strategies used by any honest party $M$.

**Definition 3.2 (Prover Strategy Assertion $P.M_{\tau}(I)$):**

$P.M_{\tau}(I)$ is true if and only if TM $M$ is a polynomial time strategy for proving $\tau(M, I)$.

One sort of a prover strategy $P.M_{\tau}(I)$ for $M$ and a given input $I$, is a strategy that has full access to all tapes of $M$. On a given input $I$, $P.M_{\tau}(I)$ is an efficient prover strategy of the knowledge of $M$ on $I$ if the tapes of $M$ can be used to generate in time polynomial in $|I|$ a proof of knowledge of $I$. The tapes can contain the proof directly or it may derive the proof through some polynomial time operation on the tape contents.

### 3.1.3 Interactive Proof of $T_{\tau}$

To provide a valid proof of $T_{\tau}(\bar{X}, \bar{Y}, I)$ for any $I$, $\bar{X}$ provides a proof part $\pi_{\bar{X}}$ and $\bar{Y}$ provides a proof part $\pi_{\bar{Y}}$. The verification of the proof of $T_{\tau}(\bar{X}, \bar{Y}, I)$ is done by computing
a polynomial function of both parts to establish the validity of $T_\tau(X, Y, I)$. Hence a proof of $T_\tau(X, Y, I)$ is a combination of a proof of $X$ and a proof of $Y$, and it is not possible for one of $X$ or $Y$ to construct a proof of $T_\tau(X, Y, I)$ independently, i.e., without interacting with the other party. A valid proof of $T_\tau(X, Y, I)$ is one that can be efficiently verified to establish that $T_\tau(X, Y, I)$ is true.

An important requirement for proofs of $T_\tau$ and their proof parts in our system is that they should not unilaterally make any explicit assertions about the knowledge of their respective machines. I.e., they should not leak the validity of the knowledge assertions $\tau(X, I)$, or $\tau(Y, I)$. This guarantees that nothing is learned beyond the validity of $T_\tau(X, Y, I)$ for a given $I$. It then follows that:

- In the case of successful verification of a proof of $T_\tau(X, Y, I)$, the assertion $T_\tau(X, Y, I)$ is established as true, which implies that $\tau(X, I)$ is true and $\tau(Y, I)$ is true.

- In the case of failed verification, only assertion $\neg T_\tau(X, Y, I)$ is established as true. I.e., no specific assertion is learned about the state of knowledge of any of the parties.

In an interaction between any TMs $Z$ and $M$ where $M$ performs the verification of $T_\tau(Z, M, I)$ on input $I$, the proof part $\pi_Z$ of $Z$ at $M$ contains the contents of the view of $M$ in the interaction $[(Z, M)(I)]$, while we assume that the proof part of $M$ is available locally to $M$. We then introduce the following requirement for proofs of $T_\tau(X, Y, I)$:

**$\tau$-Hiding Proof Requirement for $T_\tau(X, Y, I)$**: For any polynomial time strategy $M$ such that $\tau(M, I)$ is invalid, any proof parts $\pi_X$ and $\pi_Y$ should not unilaterally reveal the validity of $\tau(X, I)$ or $\tau(Y, I)$ to $M$. 

38
**Definition 3.3** (Verifier Strategy Assertion \( V.M_{T_\tau}(\pi_M, \pi_Z, I) \)):
\( V.M_{T_\tau}(\pi_M, \pi_Z, I) \) is true if and only if TM \( M \) is a polynomial time strategy for verifying the validity of \( T_\tau(M, Z, I) \) using proof parts \( \pi_M \) and \( \pi_Z \) for TM \( Z \) and input \( I \).

We introduce the following postulates for our common language computational model for all \( M, Z \) and \( I \).

**Postulate 3.1:** \( \tau(M, I) \) is valid for \( M \) and \( I \) implies that \( P.M_\tau(I) \) is true.

This postulate captures our assumption that \( M \)'s strategy for proving the validity of \( \tau(M, I) \) is of polynomial time complexity.

**Postulate 3.2:** If \( \tau(M, I) \) is valid for \( M \) and \( I \) then \( \exists \pi_M \forall \pi_Z \) such that \( V.M_{T_\tau}(\pi_M, \pi_Z, I) \) is true.

### 3.1.4 Common Language and Relation

Below we formally define the notion of a common language and the corresponding relation using \( T_\tau \) as follows:

**Definition 3.4** (\( \hat{L}_{T_\tau}(\overline{X}, \overline{Y}) \)):
The common language \( \hat{L}_{T_\tau}(\overline{X}, \overline{Y}) \) of \( \overline{X} \) and \( \overline{Y} \) is the set of \( I \) such that \( T_\tau(\overline{X}, \overline{Y}, I) \) is true.
**Definition 3.5** ($R_{LT}^L(X, Y)$): The relation $R_{LT}^L(X, Y)$ of the common language $\hat{L}_{\tau}(X, Y)$, is the set of pairs $(I, \pi)$, such that $\pi$ is a proof of the validity of $T_\tau(X, Y, I)$.

It should be noted that any proof $\pi$ of $T_\tau(X, Y, I)$ need not be computable in polynomial time. Nonetheless, there should be efficient ways for verifying $\pi$ through its proof parts. This is done through the verification strategy which uses the proof parts of $X$ and $Y$ to compute the verification function in polynomial time.

### 3.1.5 Models of Interaction on Common Languages

For any interaction $(\overline{X}, \overline{Y})(I)$ between two honest parties $\overline{X}$ and $\overline{Y}$ on $I$, each of $\overline{X}$ and $\overline{Y}$ has an efficient prover strategy for knowledge of $I$, i.e., $\tau(\overline{X}, I)$ and $\tau(\overline{Y}, I)$ are valid. The interaction between $\overline{X}$ and $\overline{Y}$ constitutes a proof that for both $\overline{X}$ and $\overline{Y}$ $\tau$ holds on $I$, i.e., that $T_\tau(\overline{X}, \overline{Y}, I)$ is true. During this interaction, both $\overline{X}$ and $\overline{Y}$ use their respective prover strategies to produce a proof for $T_\tau(\overline{X}, \overline{Y}, I)$. The proof can not be produced by $\overline{X}$ alone, similarly, it can not be produced by $\overline{Y}$ alone. The proof of $T_\tau(\overline{X}, \overline{Y}, I)$ is a function of the proof parts produced by each. The interaction between $\overline{X}$ and $\overline{Y}$ can be modeled based on their prover and verifier strategies as follows:

- **Symmetric Strategies Interaction** $(\overline{X}, \overline{Y})(I)$: In this interaction model, the prover and verifier strategies used by $\overline{X}$ and $\overline{Y}$ are identical, e.g., when each of the prover and verifier strategies at both $\overline{X}$ and $\overline{Y}$ use the corresponding TM tapes which contain identical knowledge (excluding the random tapes). This model is particularly useful for symmetric cryptographic systems.
- **Asymmetric Strategies Interaction** \((X, Y)(I)\): The prover and verifier strategies may be different at \(X\) and \(Y\), e.g., when the tapes used by the prover and verifier strategies of \(X\) and \(Y\) contain different knowledge (excluding the random tapes). This model can be used for asymmetric cryptographic systems.

We use \([(X, Y)(I)=1]\) to indicate that the two party interaction between \(X\) and \(Y\) has succeeded in the following sense: the interaction has terminated at \(Y\) and that \(Y\) was able to successfully verify that \(T_{\tau}(X, Y, I)\) is true. Note that we do not assume that \(Y\) has sent sufficient information to allow its verification at \(X\).

Consider an interaction where upon termination both \(X\) and \(Y\) have proven and verified \(T_{\tau}(X, Y, I)\). i.e., both parties were able to use their respective verification strategies to verify \(T_{\tau}(X, Y, I)\), we consider this to be a composite interaction and model it as two separate interactions \((X, Y)(I)\) and \((Y, X)(I)\). This is particularly important as we start formalizing the different components of the proposed proof system.

### 3.2 Proofs of \(T_{\tau}\) via Proofs of Knowledge leak Knowledge

In this section we consider the use of proofs of knowledge for proving \(T_{\tau}\), and we show that this approach leaks knowledge assertions \(\tau\), which contradicts the \(\tau\) hiding proof requirement for \(T_{\tau}\).

An interaction \((X, Y)(I)\) is considered a proof for \(T_{\tau}(X, Y, I)\) if it satisfies the following requirements:

- If the application specific knowledge of \(X\) and \(Y\) can be related on \(I\), then there exists a proof for \(T_{\tau}(X, Y, I)\) that can be efficiently verified.
• If one of $\overline{X}$ and $\overline{Y}$ does not have application specific knowledge for $I$, then there does not exist a valid proof for $T_\tau(X,Y,I)$, since it is invalid, and this fact can be established by an efficient verification strategy.

One approach for constructing a proof system of $T_\tau(X,Y,I)$ for any $I$, is to use proof systems of knowledge to establish the validity of $\tau(\overline{X},I)$ and $\tau(\overline{Y},I)$ on some $I$. In an interaction $(\overline{X}, \overline{Y})(I)$, $\overline{X}$ generates a proof of knowledge for $I$ at $\overline{Y}$. $\overline{Y}$ can easily validate $\tau(\overline{Y},I)$ locally, and if true, it validates the proof of knowledge of $I$ sent by $\overline{X}$. If both succeeds, then $T_\tau(\overline{X},\overline{Y},I)$ is valid, otherwise, it is invalid and the interaction $(\overline{X}, \overline{Y})(I)$, which designates a proof system for $T_\tau(\overline{X},\overline{Y},I)$ on any $I$ fails.

Nonetheless, it is important to note that the proof of knowledge used to construct a proof system for $T_\tau(\overline{X},\overline{Y},I)$ for any $I$, has to be zero knowledge, or else, the proof sent by one party $\overline{X}$ may be maliciously used by any other party $Y$ to falsely establish $T_\tau(\overline{X},\overline{Y},I)$ at possibly $\overline{X}$ or any other party, even though $\tau(Y,I)$ is false.

For example, assume that $\overline{X}$ knows a Hamiltonian cycle of some graph $G$ and that $\overline{X}$ interacts with $Y$ to assert $T_\tau(\overline{X},Y,I)$, i.e., that there exists a graph $G$, such that both $\overline{X}$ and $Y$ know a Hamiltonian cycle of $G$. If $\overline{X}$ simply sends out a cycle of $G$, then $Y$ learns a cycle of $G$ from the interaction. I.e., after any interaction $(\overline{X}, Y)(I), T_\tau(\overline{X},Y)$ trivially holds. Hence, it is important to guarantee that $Y$ learns nothing from the interaction beyond the validity of $\tau(\overline{X},I)$.

Consider an interaction $(\overline{X}, Y)(I)$ that uses ZK proofs of knowledge as described above. Before interacting with $\overline{X}$ the verifier strategy at $Y$ can easily validate $\tau(Y,I)$, and if $\tau(Y,I)$ is false, then it can trivially conclude that $T_\tau(\overline{X},\overline{Y},I)$ is invalid. Nonetheless, through its interaction with $\overline{X}$, $Y$ gains extra knowledge. Particularly, after interacting with $\overline{X}$, $Y$ learns whether $\tau(X,I)$ is true. This extra assertion is leaked knowledge that is not part of the
assertion which has been proved invalid ($\neg T_\tau(\overline{X}, \overline{Y}, I)$). This is especially important when considering $\overline{Y}$ that may interact with $\overline{X}$ even though $\tau(Y, I)$ is false.

Hence, an important requirement that distinguishes our proof system is that $\overline{Y}$ should not be able to establish that $\overline{X}$ has knowledge of $I$ solely from the proof part sent to it by $\overline{X}$. I.e., a party should only be assured of the validity of $T_\tau(\overline{X}, \overline{Y}, I)$, but should not learn that $\tau(\overline{X}, I)$ is valid by just checking the proof part sent by $\overline{X}$. It then follows that party $\overline{X}$ should not send out a verifiable proof of knowledge of $I$, i.e., the information sent out from $\overline{X}$ should not be sufficient to efficiently verify $\tau(X, I)$ on $I$. To satisfy this requirement, $\overline{X}$ must send an incomplete proof of knowledge that does not allow efficient verification of $\tau(\overline{X}, I)$ at any $Y$. The same argument applies to $\overline{Y}$. This important security feature is one of the basic differences from the standard interactive proofs of language membership or knowledge of a relation, and other variants including proofs of computational ability.

### 3.3 Interactive Proofs on Common Languages

In the previous section we have shown that existing proofs of knowledge or proofs of computational ability even in their zero knowledge form does not capture the requirement of a proof system for proving that $T_\tau(\overline{X}, \overline{Y}, I)$. The desired proof system should not make any further assertions, i.e., if $T_\tau(\overline{X}, \overline{Y}, I)$ is invalid, then the only thing that is learned from the proof system should be the invalidity of $T_\tau(\overline{X}, \overline{Y}, I)$ and no other assertions as we discussed earlier. Hence a valid proof system $T_\tau(\overline{X}, \overline{Y}, I)$ should satisfy the following requirements:

- If $\tau(X, I)$ is valid and $\tau(Y, I)$ is valid, then there exists a proof for $T_\tau(X, Y, I)$, and the system should provide a proof that can be efficiently verifiable. (I.e., there exists at least one verifier strategy).
• If \( \tau(X, I) \) is invalid or \( \tau(Y, I) \) is invalid, then \( T_\tau(X, Y, I) \) is invalid. It then follows that there does not exist a valid proof for \( T_\tau(X, Y, I) \), and this fact can be established by an efficient verification strategy of the proof system.

• If some party \( M \) does not have an efficient verification strategy \( V.M.T_\tau(\pi_M, \pi_Z, I) \)) for all party \( Z \), then \( M \) should not distinguish an honest from a dishonest \( Z \) except with negligible probability. I.e., if \( \tau(Z, I) \) is valid, \( M \) should not learn that \( Z \) has knowledge of \( I \) except if \( M \) has knowledge of \( I \). I.e., \( M \) should not distinguish \( Z \) from any other honest or dishonest parties.

### 3.3.1 Proofs of Membership in a Common Language

If an input \( I \) is in \( \hat{L}_{T_\tau}(\bar{X}, \bar{Y}) \), then \( T_\tau(\bar{X}, \bar{Y}, I) \) is valid and can be efficiently proven and verified. \( (\bar{X}, \bar{Y}) \) is a proof system of membership in the common language \( \hat{L}_{T_\tau}(X, Y) \) if it satisfies the following for all \( I \):

• If \( I \in \hat{L}_{T_\tau}(\bar{X}, \bar{Y}) \), then the interaction \( (\bar{X}, \bar{Y})(I) \) succeeds with overwhelming probability.

• If \( I \notin \hat{L}_{T_\tau}(X, Y) \) then the interaction \( (X, Y)(I) \) succeeds with negligible probability.

• For any polynomial time strategy \( D \), if \( \tau(D, I) \) does not hold, and if \( I \in \hat{L}_{T_\tau}(\bar{X}, \bar{Y}) \) then \( D \) can not distinguish \( \bar{X} \) from \( \bar{X} \) with respect to \( \tau \). Similarly, \( D \) can not distinguish \( \bar{Y} \) from \( \bar{Y} \) with respect to \( \tau \). I.e., \( D \) does not learn any better about the validity of \( \tau(\bar{X}, I) \) by interacting with \( \bar{X} \) than by interacting with \( \bar{X} \). Same argument applies to \( \bar{Y} \) and \( \bar{Y} \).

Recall that we are interested in specifying the conditions of any interaction that involves at least one honest party. Interactions in which both parties can be dishonest are not of interest as both can exhibit arbitrary behaviors. Moreover, we focus on the “yes” instances
Summary of Postulates of the Common Language Model:

- **Postulate 3.1**: If \( \tau(M, I) \) is valid for \( M \) and \( I \) then \( P.M.\tau(I) \) is true.

- **Postulate 3.2**: If \( \tau(M, I) \) is valid for \( M \) and \( I \) then \( \exists \pi_M \forall \pi_Z \text{ such that } V.M.T.\tau(\pi_M, \pi_Z, I) ) \) is true.

Table 3.2: Common Languages Model Postulates.

... of the interactions, i.e., ones in which the interactions succeed. We now formally define interactive proofs of membership in a common language in Table 3.2.

**Example**: To demonstrate the notion of proofs on common languages, consider a locked secret room that can only be accessed via one of two locked doors \( A \) and \( B \), each with a unique key. Once inside the room, parties can make agreements, and engage in important conversations of interest based on the established trust of unlocking the same room.

Consider parties \( X \) and \( Y \) that do not trust each other. Party \( X \) owns a key that unlocks door \( A \), while party \( Y \) owns a key that unlocks door \( B \). Both \( X \) and \( Y \) can then unlock the same room, even though they have different keys (Completeness), and following our interaction model, assume that \( Y \) is the party that will acknowledge the success of the meeting with \( X \).

A party that does not have a valid key to any of the two doors should not be able to unlock the room and meet anyone inside (Prover Soundness). Similarly, a party should not be able to determine who is capable of unlocking the room either, as it can not go inside the room to check, which is the only way to know if someone can actually unlock the room, as nobody is willing to reveal his key or use it in front of others (Verifier Soundness). A
Definition 3.6 (Interactive Proof of Membership in a Common Language) A pair of interactive polynomial time probabilistic Turing machines \((\mathcal{X}, \mathcal{Y})\) is an “Interactive Proof System of Membership in a Common Language \(\mathcal{L}_{\tau}(\mathcal{X}, \mathcal{Y})\)” if there exists a negligible function \(\varepsilon : \mathbb{N} \rightarrow [0, 1]\) such that:

- Completeness: \(\forall I : (I \in \mathcal{L}_{\tau}(\mathcal{X}, \mathcal{Y}) \Rightarrow \Pr[(\mathcal{X}, \mathcal{Y})(I) = 1] \geq 1 - \varepsilon(|I|))\).
- Prover Soundness: \(\forall I \forall X : (I \notin \mathcal{L}_{\tau}(\mathcal{X}, \mathcal{Y}) \Rightarrow \Pr[(\mathcal{X}, \mathcal{Y})(I) = 1] \leq \varepsilon(|I|))\).
- Verifier Soundness: Let \(\Pr^X_D(\tau(\mathcal{X}, I))\) denote the probability that polynomial time strategy \(D\) outputs 1 for \(\tau(\mathcal{X}, I)\) after black box interaction with \(\mathcal{X}\). \(\Pr^Y_D(\tau(\mathcal{Y}, I))\) is defined similarly for \(\mathcal{Y}\). Then \(\forall I \forall X \forall Y \forall D\) such that \((I \in \mathcal{L}_{\tau}(\mathcal{X}, \mathcal{Y}) \land \neg \tau(D, I))\):

\[
(\left| \Pr^X_D(\tau(\mathcal{X}, I)) - \Pr^Y_D(\tau(\mathcal{Y}, I)) \right| \leq \varepsilon(|I|)) \land (\left| \Pr^X_D(\tau(\mathcal{X}, I)) - \Pr^Y_D(\tau(\mathcal{Y}, I)) \right| \leq \varepsilon(|I|))
\]

Table 3.2: Interactive Proof of Membership in a Common Language.

desirable property that will be discussed in the next section is that once inside the room, no party should leak anything about its key to others (Zero Knowledge). We make the following remarks on the proposed definition:

- Prover soundness focuses on party \(X\), i.e., the party that is not expected to verify \(T_\tau(X, Y, I)\), but only to provide its proof part for \(T_\tau(X, Y, I)\). This soundness requirement prevents a dishonest party \(\tilde{X}\) from impersonating an honest party \(X\), i.e., from falsely claiming that it can prove knowledge of \(I\).

- Verifier soundness guarantees that the \(\tau\) hiding requirement holds. We call it verifier soundness because it controls the verification of the knowledge assertion \(\tau\) for parties \(X\) and \(Y\). It considers indistinguishability relative to any polynomial time strategy \(D\) for the knowledge assertion \(\tau\) of \(\overline{X}\) and \(\overline{Y}\). This requirement guarantees that no
polynomial time strategy $D$ can learn the validity of $\tau$ through an interaction with $\overline{X}$ or $\overline{Y}$ in the proof system. Verifier soundness is formulated using computational indistinguishability that makes use of black box interactions with honest party $\overline{X}$ and any party $X$. $D$ is allowed to interact with $\overline{X}$, which is following its program for proving $T_\tau$, and the interaction may involve possible resetting of $\overline{X}$. Similarly $D$ is allowed to interact with $X$, which may be dishonest. In each interaction, $D$ should terminate by outputting a bit (1/0) indicating if $\tau$ is valid (1) or not valid (0), i.e., if the party with which it was interacting has knowledge of $I$ or not. The difference between the probability that $D$ outputs 1 when interacting with $\overline{X}$, and when interacting with any $X$ on $I$ should be negligible. The same requirement applies for $D$ in its experiment with $\overline{Y}$ and $Y$. It is important to note that we do allow $D$ to use $\overline{Y}$ in its experiment with $\overline{X}$, and we allow $D$ to use $\overline{X}$ in its experiment with $\overline{Y}$. The use is limited to black box access though, i.e., $D$ can not access the tapes of $\overline{X}$ or $\overline{Y}$.

### 3.3.2 Zero Knowledge Proofs on Common Languages

A zero-knowledge proof allows a prover to convince a verifier of an assertion without revealing any further information beyond the fact that the assertion is true. Hence, the zero knowledge property protects a prover strategy against knowledge leakage as a result of a proof system interaction.

For proofs on Common Languages, each of the parties $\overline{X}$ and $\overline{Y}$ involved in the interaction have efficient prover strategies, i.e., it is no longer the case that knowledge can be leaked from only one party, the prover. In the new proof system, however, both parties can make use of their prover strategies during an interaction, hence it is important to guarantee
that not only that no knowledge is leaked through a proof interaction with \(X\), but also that no knowledge is leaked through a proof interaction with \(Y\).

The simulation paradigm has been central to formulating the zero knowledge property, its different types and variants. A simulator is an algorithm that tries to simulate the interaction of an adversary with an honest party, without knowing the private input of this honest party. Traditional zero knowledge proofs focus on ensuring that \(Y\) hasn’t learned anything about \(X\)’s knowledge as the result of its interaction with \(X\). This is captured by ensuring that \(Y\) could have simulated the entire interaction by itself. Therefore, it has gained no further knowledge as a result of interacting with \(X\), beyond what it could have discovered by herself given the input \(I\).

For proofs on common languages, the simulator has to be used in both directions. Hence, a simulator is required for \(X\) to guarantee that any strategy \(\hat{C}\) learns nothing from interacting with \(X\) by ensuring that \(\hat{C}\) can simulate the whole interaction with \(X\) by itself. Similarly, a simulator is needed for \(Y\) to guarantee that any strategy \(\hat{C}\) learns nothing from interacting with \(Y\) by ensuring that \(\hat{C}\) can simulate the whole interaction with \(Y\) by itself. Note that \(\hat{C}\) can be any strategy, including ones that has knowledge on the common input. Our zero knowledge formulation for proofs on common languages is given below for computational zero knowledge, and can be similarly defined for other variants of zero knowledge.

**Definition 3.7 (Computational Zero Knowledge for Proofs on Common Languages)** Let \((X, Y)\) be an interactive proof system for common language \(\hat{L}_{T_{\tau}}(X, Y)\), let \(\hat{C}\) be any arbitrary polynomial time strategy. Denote by \(\{(X, \hat{C})(I)\}\) the probability ensembles of all the read-only tapes \((VIEW_{\hat{C}})\) of \(\hat{C}\), when interacting with \(X\) on common input \(I \in \hat{L}_{T_{\tau}}(X, Y)\),
and denote by \{ (\hat{C}, \overline{Y})(I) \} the probability ensembles of all the read-only tapes (VIEW_{\hat{C}}) of \hat{C}, when interacting with \overline{Y} on common input \( I \in \hat{L}_{T_{\hat{C}}} (\overline{X}, \overline{Y}) \). We say that the proof system \((\overline{X}, \overline{Y})\) is Computational Zero-Knowledge (for \( \hat{L}_{T_{\hat{C}}} (\overline{X}, \overline{Y}) \)) if and only if the following two conditions hold for all expected polynomial-time TM \( \hat{C} \):

- There exists an expected polynomial-time simulator \( M_{\hat{C}} \), such that the probability ensembles \( \{ M_{\hat{C}}(I) \}_{I \in \hat{L}_{T_{\hat{C}}} (\overline{X}, \overline{Y})} \) and \( \{ (\hat{C}, \overline{Y})(I) \}_{I \in \hat{L}_{T_{\hat{C}}} (\overline{X}, \overline{Y})} \) are polynomially indistinguishable.

- There exists an expected polynomial-time simulator \( M_{\hat{C}} \), such that the probability ensembles \( \{ M_{\hat{C}}(I) \}_{I \in \hat{L}_{T_{\hat{C}}} (\overline{X}, \overline{Y})} \) and \( \{ (\overline{X}, \hat{C})(I) \}_{I \in \hat{L}_{T_{\hat{C}}} (\overline{X}, \overline{Y})} \) are polynomially indistinguishable.

### 3.3.3 Zero Knowledge Proofs on Common Languages: Efficient Secure Two Party Computation for Authentication

Zero knowledge proofs on common languages constitute a simple and secure two party computation for authentication that is resilient to server breaches and enjoy a number of security properties, including privacy and anonymity.

To gain some insight into zero knowledge proofs on common language, let us first recall the classic story of Ali Baba’s strange cave that is commonly used to demonstrate the concept of zero knowledge interactive proof systems [63]: A cave entrance forks in two paths that deadend at a shared magic wall as shown in Figure 3.1. The wall can be made to disappear at the deadend of one path by uttering a secret word, to reveal the other path. Now Peggy knows the secret word and Victor does not. Peggy can prove to Victor that she knows the secret without revealing it as follows: Peggy enters the cave first, randomly choosing a path, and traverses it till she deadends into the magic wall. Victor then randomly choose
a path and shout to Peggy to return from that path. If Peggy does not know the secret, she gets lucky by having traversed the same path that Victor later chooses (and thus fools Victor into believing she knows the secret) only with a probability of half. But if she knows the secret, she can always convince Victor that she knows the secret.

Now consider a variation on the story for zero knowledge interactive proofs on common languages: Peggy and Victor each knows a secret, possibly different, word that opens the magic wall. We also assume a modified cave, where the two paths are also connected by a path other than the one blocked by the magic wall (as shown in Figure 3.2). Victor waits outside the cave and give Peggy a note on a piece of paper. Peggy randomly chooses a way down to the magic wall, utters the secret word and opens the wall. She then leaves the note underneath the wall before it closes again. Peggy then gets out of the cave to meet Victor (she can even use the connector path to get out on the same path she used to get in). Victor then comes in the cave down to the magic wall, utters his secret word to open the wall and verify that the note is underneath the door. He then picks up the note, hide it, and take any random path to get out of the cave. Clearly If Victor does not know the wall’s secret, he can never determine if Peggy is honest or cheating. Similarly, a cheating Peggy, can
not succeed in fooling an honest Victor who knows the secret word. It is only when both Peggy and Victor know the secret word that any of them learns that the other indeed knows a secret word.

In the original cave story, Victor authenticates Peggy based on its knowledge of the secret word that opens the magic wall. In this case, there is no assumptions on Victor’s knowledge. Hence, the knowledge of the secret word can be used as an identifying fact for Peggy.

On the other hand, in the modified cave story, we assume that both Peggy and Victor knows some secret word, hence, trust can be established on the fact that both Peggy and Victor can prove knowledge of the wall secret. In this example, Victor performs the check, but its always possible for Peggy to do her check as well (simply Peggy can go back to the wall later and check that Victor has indeed picked up her note from under the wall).

The modified cave experiment can be seen as a secure two party protocol for the following joint authentication computation $f$: “If both Peggy and Victor knows the secret word return 1 to both, otherwise return 0 to both”. It is then required to derive a protocol which allows Peggy and Victor to provide their inputs and learn the output of $f$. To achieve a secure computation of $f$, both Peggy and Victory should learn nothing beyond their inputs and the output of $f$. Following the ideal/real world paradigm for security analysis of two party computations, an ideal implementation involves a trusted third party securely verifying that both Peggy and Victor knows a secret word, and then sending 1 to each if both knows the secret word, and 0 otherwise. In the ideal world, Peggy and Victor knows nothing about each others state of knowledge of the wall secret, but are only allowed to learn the respective output of the authentication computation. As an informal security argument for the protocol discussed above, we note that if Victor does not know the secret word, it
has no way of knowing if Peggy knows a secret word, since the cave has a connector path, and any party can traverse the path completely skipping the wall. Similarly, if we assumed that Peggy should verify the authentication function as well, then if Peggy does not know the secret word, it can not learn if Victor knows the secret word or not, since it has no way of checking if Victor has tried to open the secret wall and check for the presumed note.

3.4 Multiprover Proofs of Membership in a Common Language

We generalize proofs on common languages to have several \((k)\) \(X\) parties that cooperate to convince the same party \(Y\) of the same thing, the membership of a given input \(I\) in a common language of the \(k\) \(X\) parties and party \(Y\). Following the traditional multiprover interactive proof systems [15], the \(k\) parties are not allowed to communicate with each other or see each other’s exchanges with \(Y\). This is similar to the interrogation of criminals where each suspect is interrogated separately and without allowing them to hear what the others have told the police. Each of the \(k\)-pair interactions \((X_i, Y)\) for all \(i \in [1, k]\) designate a prover contributing to the proof of membership in the common language \(\hat{L}_{\Pi}(\langle X_1, X_2, ..., X_k \rangle, Y)\).
Definition 3.8 (Multiprover Interactive Proof of Membership in a Common Language) A set of $k$ polynomial time probabilistic Turing machines $X_i, i \in [1,k]$, interacting with polynomial time probabilistic Turing machine $Y$, is a “$k$-prover Interactive Proof System of Membership in a Common Language $\hat{L}_{T_\tau}((\overline{X}_1,\overline{X}_2,..,\overline{X}_k), Y)$ ” if there exists a negligible function $\varepsilon : N \rightarrow [0,1]$ such that:

- Completeness: $\forall I: (I \in \hat{L}_{T_\tau}((\overline{X}_1,\overline{X}_2,..,\overline{X}_k), Y): \Pr[((\overline{X}_1,\overline{X}_2,..,\overline{X}_k), Y)(I)=1] \geq 1 - \varepsilon(|I|))$.

- Prover Soundness: $\forall I \forall X_i \forall i \in [1,k]: (I \notin \hat{L}_{T_\tau}((X_1,X_2,..,X_k), Y): \Pr[((X_1,X_2,..,X_k), Y)(I)=1] \leq \varepsilon(|I|))$.

- Verifier Soundness: Let $Pr^{X_i}_{D}(\tau(X,I))$ denote the probability that polynomial time strategy $D$ outputs 1 for $\tau(X_i,I)$ after black box interaction with $X_i$. $Pr^{Y}_{D}(\tau(Y,I))$ is defined similarly for $Y$. Then:

$$\forall I \forall X_i \forall i \in [1,k] \forall Y \forall D: (I \in \hat{L}_{T_\tau}((\overline{X}_1,\overline{X}_2,..,\overline{X}_k), Y) \land \neg \tau(D,I): (|Pr^{X_i}_{D}(\tau(X,i,I)) - Pr^{X}_{D}((\overline{X}_i,I))| \leq \varepsilon(|I|)) \land (|Pr^{Y}_{D}(\tau(Y,I)) - Pr^{Y}_{D}(\tau(Y,I))| \leq \varepsilon(|I|)))).$$

Table 3.3: Multiprover Interactive Proof of Membership in a Common Language.

We formally define multiprover interactive proofs of membership in a common language in Table 3.3.

3.5 Discussion

In this section, we discuss some issues related to the common language model and proofs on common languages, and introduce the next chapter.

3.5.1 Common Languages based on Common Computational Ability

The assertion $T_\tau(X,Y,I)$ can be defined using common computational ability instead of common knowledge of $I$. I.e., $\tau(M,I)$ is then defined in terms of computational ability as follows: “$M$ has computational ability for $I$, and the validity of the assertion can be
established in polynomial time”. The prover strategy assertion $PM_\tau(I)$ is then true if $M$ can prove computational ability on $I$. For example, when the tapes of $M$ may not contain knowledge that can be used directly to generate the proof, $M$ proves computational ability by using some black box oracle as a helper. In this case, the prover strategy assertion is true if $M$ can use the helper oracle to prove that $M$ has computational ability for $I$. Then, $T_\tau(X,Y,I)$ denotes an assertion on common ability, which assert that $PX_\tau(I)$ is true and $PY_\tau(I)$ is true for parties $X$, $Y$ and input $I$, i.e., that $X$ can prove that it has computational ability for $I$ and $Y$ can prove that it has computational ability of $I$.

### 3.5.2 Verification of $T_\tau(X,Y,I)$

The validity of $V.M_{T_\tau}(\pi_X,\pi_Y,I)$ implies that $M$ can efficiently verify $T_\tau(X,Y,I)$, using proof parts from both $X$ and $Y$ to compute a polynomial time verification function. Nonetheless, the proof for $T_\tau(X,Y,I)$ in the form used in the common language relation is not restricted to ones that can be produced in polynomial time. Proofs on common languages uses the fact that both $X$ and $Y$ are knowledgeable on $I$ to simplify the proofs that $X$ and $Y$ can prove knowledge of $I$. This is demonstrated in our protocol suite presented in Chapters 5 and 6.

### 3.5.3 Designing Common Languages

In the next chapter, we detail the steps for designing common languages. This is the first step towards building protocols for realizing proofs on common languages. We demonstrate our approach with examples based on NP and co-NP languages. The examples demonstrate that $\tau(M,I)$ for TM $M$ and input $I$ need not be as assertion on the knowledge that $I$ is in some language. The common language can be a subset of other languages, or it can be defined independently on any known languages. For example, in the next chapter we
propose a common language design for Discrete Log that allows for non identical provers. The members of the common language \( I = \langle \ell, w, g_x, g_y, q, p \rangle \) using generators \( g_x \), and \( g_y \) and values \( \ell \) and \( w \), is not a member in the DL language, however, the common knowledge assertion on \( I \) is formulated in terms of membership in other languages, including NP, co-NP, and possibly other common languages as we demonstrate in the next chapter. The designs proposed in next chapter suggests that the set of languages provable by proofs on common languages include those provable by interactive proofs (IP), but we have not concluded if there is equivalence between these sets, or if the class of languages provable by proofs on common languages can span languages beyond class IP.
Chapter 4: Protocol Design for Proofs on Common NP Languages

In Chapter 3 we introduced the common language model, and proposed a proof system for proving membership in common languages. In this chapter we detail our approach, design and analyze the common languages which we will use for the protocols in the following chapters. We derive common languages for various NP languages, as a first step towards building protocols constituting proofs of membership in these common languages. We demonstrate that common languages can be derived for languages in co-NP as well. Through our co-NP example we demonstrate the possibility of nesting common languages, by defining one in terms of the other, later in the chapter. Recall that the common languages design involves defining assertion $\tau$, $T_\tau$ and the underlying relation for the common language. Specifically, we study the design of common languages for the languages of Discrete Log (DL), RSA, Graph Hamiltonian (HML), and our last example is the co-NP Graph non Isomorphism (NOI). The proposed designs satisfy the common language model requirements specified by the postulates in Table 3.1.

4.1 Our Approach

Below we summarize the steps involved in designing common languages based on any abstract language. The design can make restrictions on various levels specifying the types
of commonalities allowed in the common language. Hence, for a given abstract language, there can be different designs for common languages based on the specified constraints.

• **Select a Language:** Given two parties $\overline{X}$ and $\overline{Y}$ select a computationally hard problem, e.g., finding the prime factors of a composite number, for which there are instances that can be efficiently solved using the knowledge of $\overline{X}$ and instances that can be efficiently solved using the knowledge of $\overline{Y}$. We shall denote this hard problem as $AP$. If the knowledge at $\overline{X}$ is different than the knowledge at $\overline{Y}$, ensure that $AP$ allows for non identical instances to be solvable by the knowledge of $\overline{X}$ and the knowledge of $\overline{Y}$. It is then important to figure out language specific ways of relating the non identical knowledge at $\overline{X}$ and $\overline{Y}$. In this step, an abstract language, $L_A$, is defined, i.e., the set of values for which providing a proof involves solving the hard problem $AP$. Hence, $\overline{X}$ has knowledge of a set of values in $L_A$, and $\overline{Y}$ has knowledge of a set of values in $L_A$.

• **Specify the knowledge assertion $\tau(M,I):$** The concrete common language $L_C$ of two parties, constitutes a refinement of the selected abstract language $L_A$. The knowledge of the two parties on values of $L_A$ is mapped through a simulation relation to the values of $L_C$. The knowledge assertion for party $M$ on value $I$ of the concrete common language $L_C$ is then specified in terms of the knowledge of the values of $L_A$ for which there is a simulation mapping to $I$.

• **Derive the Common Language Simulation:** To define the simulation relation between $L_A$ and $L_C$, start by selecting another hard problem $SP$, such that an instance of $SP$ is efficiently solvable only if specific instance(s) of $AP$ is efficiently solvable. For example, if $AP$ is the hardness of finding the Discrete Log, then $SP$ can be the
intractability of the computational Diffie-Hellman problem, and a party that knows
the discrete log of some $I'$ in $L_A$ can efficiently derive the Diffie-Hellman key (DH)
that uses key parts $I'$ and any other random key part $R$. It then follows that the knowl-
edge of the discrete log enabled the party to solve the instances of the Computational
Diffie Hellman problem, in which $I$ is a key part. Hence, a solver for $SP$ instances
can be constructed using $\overline{X}$, and the same applies for $\overline{Y}$, since both have knowledge
for some instances of $AP$. The simulation relation then uses a solver strategy for $SP$
instantiated using $\overline{X}$ and using $\overline{Y}$ to create two solvers whose output can be related
by an equivalence relation or any other efficiently checkable relation based on the
design. This captures the requirement that both parties use their knowledge, that
may not be identical, to do the same things, or solve the same problems, hence the
common language. The following conditions must hold for the simulation relation:

- For every value in $L_C$ there is at least one value in $L_A$ that maps to it through the
  simulation relation.

- For every value in $L_C$, the values in $L_A$ that are mapped onto it should include
  at least one value for which $\overline{X}$ has knowledge, and one value for which $\overline{Y}$ has
  knowledge.

- The simulation mapping has to guarantee that the hardness assumptions $AP$ and
  $SP$ are maintained.

**A note on the solvers:** We use the assertion $T_\tau(\overline{X}, \overline{Y}, I)$ to construct efficient solvers for
hard problems, one based on $\overline{X}$ and the other based on $\overline{Y}$. We define the following two
constraints on the solvers; first, the solvers have to solve all instances of $SP$ that guarantee
that the knowledge of $\overline{X}$ and $\overline{Y}$ is used. For example, for the hard problem of finding the
Diffie-Hellman Key, the solvers should be defined for all instances that include $I$ as a DH key part (we shall elaborate on the specifics of $I$ that are used for the DH key part in the next section). The universal quantification guarantees that no accidental knowledge of another key part will be used to falsely prove the assertion $T_\tau(X,Y,I)$. Second, the output of the solvers for all these instances has to be related in some way, e.g., in our proposed common languages designs, we use solvers that produce identical values. To simplify our discussion we use the output of the solvers as the proof parts, nonetheless, it should be noted that the proof parts may contain randoms and other values beside the solvers outputs. We assume that there exists efficient ways for an honest prover strategy to construct a proof part using the solver output, and that there exists efficient ways for an honest verification strategy to extract the solver output from a proof part.

4.2 Common Languages for Discrete Log

If $G$ is a group, such as the multiplicative group of a finite field or the group of points on an elliptic curve, and $g$ is an element of $G$, then $g^n$ is the discrete exponentiation of base $g$ to the power $n$. The inverse operation is, given $w$ in $G$, to find a value for $n$, if it exists, such that $w = g^n$, where $n \in [0, |\langle g \rangle|]$, and $\langle g \rangle$ is the subgroup generated by $g$. This defines the problem of finding the discrete logarithm of $s$ to base $g$, which is one of the important problems of cryptography that are known to be hard, especially for some groups, such as those from elliptic curves.

The language of Discrete Log ($DL$) is known to be in the computational complexity class NP, specifically in class NP-intermediate, i.e., it is not known to be in P, nor in NP-complete. Assume a multiplicative group modulo a prime $p$, and a subgroup of order $q$ of
DL language and relation:

- $R_{DL} = \{(\langle w, g, q, p \rangle, s) : g^s = w \pmod{p}\}$.

- $DL = \{\langle w, g, q, p \rangle : (\exists s \in [0, q-1] : (\langle w, g, q, p \rangle, s) \in R_{DL})\}$.

Table 4.1: NP Language of Discrete Log.

the multiplicative group modulo $p$ where $q|\,(p-1)$. The $DL$ language and its underlying relation are given in Table 4.1 (language member $I$ is given by the vector enclosed in $\langle \ldots \rangle$).

For every vector $I = \langle w, g, q, p \rangle$ in $DL$, the discrete log $s$ of $w$ constitutes a proof that $\langle w, g, q, p \rangle$ is in $DL$. Nonetheless, for any $I$, $s$ trivially exists, i.e., each $I$ is in $DL$. This led earlier proof systems for the Discrete Log problem to focus on verifying that the discrete log $s$ of a given $w$ is known, rather than the trivial fact that it exists, i.e., the assertion $\tau$ was made on the knowledge of the prover on a given input vector $I = \langle w, g, q, p \rangle$ rather than the membership of $I$ in $DL$.

Traditional proof systems of knowledge, generalized as proofs of computational ability, make use of efficient knowledge extractor strategies for establishing its soundness requirement. Unfortunately, this made such proof systems by definition vulnerable to many attacks, for example resettable attacks. Here we show that it is possible to construct non-trivial efficient and secure proof systems for this important NP language using proofs on common languages, where membership assertions are made instead of knowledge assertions.

Many possible common relations can be defined to relate two parties $X$ and $Y$ on $DL$. Below we give two examples on common languages and their associated common relations.
both based on the abstract language $DL$. The first one is restricted to identical prover strategies at $X$ and $Y$, while the second one demonstrates a broader common relation that allows $X$ and $Y$ to have different prover strategies based on knowledge of different discrete logs.

The idea is to use the knowledge of the discrete log of $I$, and to find a common value that both $X$ and $Y$ can produce using their respective knowledge of the discrete log of $I$. This provides a basis for defining the relation of the common language. In doing so, we let the common value computation exploit the intractability of the Computational Diffie Hellman problem (CDH). Computing a Diffie-Hellman key from two key parts $k_1$ and $k_2$ is known to be hard lest the discrete log of at least one of the key parts is known. Hence, if both $X$ and $Y$ have efficient prover strategies for one of the key parts, then both should be able to compute the same Diffie-Hellman key (DH) regardless of the value of the other key part.

Hence, if $T_I(X,Y,I)$ is true, then there exist an efficient solver strategy for the CDH problem that can be instantiated for $X$ and for $Y$ to create two CDH solvers based on the prover strategies of these parties. These solvers can solve the CDH problem for instances of $CDH$ for which one of the key parts is $I$. We quantify over all other key parts to guarantee that no accidental knowledge of another key part, other than $I$, is used in computing the common DH key. If $I$ is in the common language, then both solvers should produce the same Diffie-Hellman key $DH(I,k_2)$, for key parts $I$ and any other key part $k_2$.

A protocol implementing a proof system on this common language need to verify that each of $X$ and $Y$ can compute any DH key for which $I$ is a key part. Hence, it is important
to guarantee that neither X nor Y know the discrete log of the other key part and that the CDH assumption holds.

**A note on identical versus non-identical prover strategies:** In the following common language designs, for identical knowledge of discrete logs, i.e., to the same base, generator $g$, we do a universal quantification over all key parts $w'$, whereas for non identical knowledge of discrete log, i.e., to different bases, $g_x, g_y$, the universal quantification is done over all the exponents for the key parts, which still implies universal quantification over all key parts, but allow us to more precisely specify the common DH computation.

### 4.2.1 DL Common Language for Identical Prover Strategies

We start by identifying theorems $DL1-\tau$, $DL1-T_\tau$, the common $DL$ language $DL1-\hat{L}_{T_\tau}(X,Y)$ and the common relation $DL1-R_{L_\tau}(X,Y)$ as listed in Table 4.2. The members of the common language is of the form $I = \langle w, g, q, p \rangle$. The assertion $\tau$ asserts that a Turing Machine $M$ has knowledge of the discrete log of $w$ to the base $g$. CDH-DL1 denotes the CDH solver strategy that uses the discrete log problem specific knowledge of a given party to solve the CDH problem for any CDH instances that uses $w$ as one of its key parts.

Such solvers are made only possible through the knowledge of $X$ and $Y$ on $w$, otherwise, and with the assumed hardness assumption of CDH, such solvers do not exists or else they can be used to break the intractability of CDH. If $V.Y_{T_\tau}(\pi_Y, \pi_X, I))$ is true, then $Y$ can check that the output of the two solvers are equal. In this case a proof part $\pi_X$ provides the DH key produced by the solver based on $X$ for key parts $w$ and $w'$. Similarly, the output of the solver based on $Y$ is provided by $\pi_Y$, which is the DH key produced for the key parts $w$ and $w'$. The verification is an efficient equality check done on the two DH keys.
DL common language and common relation for parties $X$ and $Y$ designed for identical prover strategies.

- $DL1-R_{L\tau}(X, Y) = \{ \langle w, g, q, p \rangle, CDH-DL1 \} : (\forall w' (CDH-DL1(X, g, w', w, q, p) = CDH-DL1(Y, g, w', w, q, p) = k) \land (k \text{ is the Diffie-Hellman key of key parts } (w, w') \text{ for generator } g)) \}.$

- $DL1-\hat{L}_{T\tau}(X, Y) = \{ \langle w, g, q, p \rangle : (\exists CDH \text{ solver } CDH-DL1((\langle w, g, q, p \rangle, CDH-DL1) \in DL1-R_{L\tau}(X, Y))) \}.$

- $DL1-\tau (M, \langle w, g, q, p \rangle): M \text{ knows } s \text{ such that } (\langle w, g, q, p \rangle, s) \text{ in } R_{DL}.$

- $P.M_\tau(\langle w, g, q, p \rangle): M \text{ is an efficient strategy for proving } DL1-\tau (M, \langle w, g, q, p \rangle).$  

- $DL1-T_{\tau}(X, Y, \langle w, g, q, p \rangle): DL1-\tau(X, \langle w, g, q, p \rangle) \land DL1-\tau(Y, \langle w, g, q, p \rangle).$

---

Table 4.2: Specification of Theorems $DL1-\tau$ and $DL1-T_{\tau}$, and the Common Language $DL1-\hat{L}_{T\tau}(X, Y)$.
Theorem 4.1  The common language $DL1-\hat{L}_\tau(X,Y)$ satisfies the requirements of the common language model.

Proof  We show that $DL1-\hat{L}_\tau(X,Y)$ satisfies the postulates of the model listed in Table 3.1.

- **Postulate 3.1:** Based on the definition of $\tau(M,I)$ in 4.2, $M$ knows the discrete log of $w$, hence, $M$ can prove that it has knowledge by simply outputting $s$, the discrete log of $w$, which can be done in polynomial time, since $M$ knows $s$.

- **Postulate 3.2:** If $X$ is a prover for $\tau(X,I)$, then it can output $s$ such that $g^s = w$, hence it can efficiently compute the DH key for key part $w$ and any other key part $w'$ by deriving $\pi_X = w'^s$. Hence for any $Y$ and $\pi_Y$, $X$ can efficiently establish if $I$ is in the common language by comparing $\pi_X$ to $\pi_Y$ for any random $w'$, if they are equal, then $DL1-T_\tau(X,Y,\langle w,g,q,p \rangle)$ is true, otherwise, it is false. Similarly, if $Y$ is a prover for $\tau(Y,I)$, then it can output $s$ such that $g^s = w$, hence it can efficiently compute the DH key for key part $w$ and any other key part $w'$ by deriving $\pi_Y = w'^s$. Hence for any $X$ and $\pi_X$, $Y$ can efficiently establish if $I$ is in the common language by comparing $\pi_X$ to $\pi_Y$ for any random $w'$, if they are equal, then $DL1-T_\tau(X,Y,\langle w,g,q,p \rangle)$ is true, otherwise, it is false. It then follows that if $P.M_\tau(I)$ is true for TM $M$ (can be $X$, or $Y$) then $\exists \pi_M \forall \pi_Z$ such that $V.M_\tau(\pi_M,\pi_Z,I))$ is true.

- **$\tau$-Hiding Proof Requirement for $T_\tau(X,Y,I)$:** Based on the intractability of the CDH assumption, there exists no DH solver that can output the DH key given only
the key parts without knowledge of the discrete log of any of the key parts. It then follows that it should be hard to compute the CDH key for the key part \( w \) and any key part \( w' \) based on the hardness of static CDH (which is based on the CDH assumption but assumes that one of the key parts is fixed or static, while the other changes). Hence, if \( M \) can output a DH key constituting \( \pi_M \) for all \( \pi_Z \) on all key parts \( w' \), then \( M \) is a solver, but since a solver does not exists, then it must be that \( M \) knows the discrete log of \( w \), i.e. that \( P.M(\langle w, g, q, p \rangle) \) is true. It then follows that if the prover assertion does not hold for \( M \), i.e., \( \tau \) is false for \( M \) and a given \( I \), then it should not be able to establish if \( \tau(Z,I) \) holds or not for any \( Z \), since it can not efficiently derive the necessary DH key.

4.2.2 DL Common Language for non Identical Prover Strategies

Next, we propose a different common language structure that allows \( X \) and \( Y \) to have different prover strategies. Table 4.3 lists theorems \( DL2-\tau \), \( DL2-T_\tau \), the common \( DL \) language \( DL2-\hat{L}_{T_\tau}(X,Y) \) and the common relation \( DL2-R_{\hat{L}_{T_\tau}(X,Y)} \). The members of this language is of the form \( I=\langle \ell, w, g_x, g_y, q, p \rangle \). The assertion \( \tau \) asserts that a Turing Machine \( M \) has knowledge of the discrete log of \( w \) to the base \( g_x \) or base \( g_y \). CDH-DL2 denotes the CDH solver strategy that uses the discrete log problem specific knowledge of a given party to solve the CDH problem for any CDH instances that uses \( w \) as one of its key parts.

Such solvers are made only possible through the knowledge of \( X \) and \( Y \) on \( w \), otherwise, and with the assumed hardness assumption of CDH, such solvers do not exists or else they can be used to break the intractability of CDH. If \( V.Y_{T_\tau}(\pi_Y, \pi_X, I)) \) is true, then \( Y \) can check that the output of the two solvers are equal. In this case a proof part \( \pi_X \) is the DH
Table 4.3: Specification of Theorems \( DL2, \tau \) and \( DL2-T \), and the Common Language \( DL2\hat{-}T \).

key produced by the solver based on \( X \) for key parts \( w \) and \( w' \). Similarly, the output of the solver based on \( Y \) constitutes the proof part \( \pi_Y \) which is the DH key produced for the key parts \( w \) and \( w' \). The verification is an efficient equality check done on the two proof parts.

**Theorem 4.2** The common language \( DL2\hat{-}T_{\hat{t}}(X, Y) \) satisfies the requirements of the common language model.

**Proof** We show that \( DL2\hat{-}T_{\hat{t}}(X, Y) \) satisfies the postulate of the model listed in Table 3.1.
• **Postulate 3.1:** Based on the definition of $\tau(M, I)$ in 4.3, $M$ knows the discrete log $s$ of $w$ to base $g_y$ or to base $g_x$, such that $g_x^s = w$ or $g_y^s$, hence, $M$ can prove that it has knowledge by simply outputting the corresponding discrete of $w$, which can be done in polynomial time.

• **Postulate 3.2:** If $X$ is a prover for $\tau(X, I)$, then it can output $s$ such that $g_x^s = w$ or $g_y^s = w$, hence it can efficiently compute the DH key for key part $w$ and any other key part $w'$ or $w''$ such that the discrete log of $w'$ to base $g_y$ is the same as the discrete log of $w''$ to base $g_x$. The DH key is then computed as $\pi_X = w'^s$ if $s$ is the discrete log to base $g_y$, or $\pi_X = w''^s$ if $s$ is the discrete log to base $g_x$. Hence for any $Y$ and $\pi_Y$, $X$ can efficiently establish if $I$ is in the common language by comparing $\pi_X$ to $\pi_Y$, if they are equal for $w'$ and $w''$, then $DL2-T_\tau(X, Y, (\ell, w, g_x, g_y, q, p))$ is true, otherwise, it is false. Similarly, if $Y$ is a prover for $\tau(Y, I)$, then it can output $s$ such that $g_x^s = w$ or $g_y^s = w$, hence it can efficiently compute the DH key for key part $w$ and any other key part $w'$ or $w''$ such that the discrete log of $w'$ to base $g_y$ is the same as the discrete log of $w''$ to base $g_x$. The DH key is then computed as $\pi_Y = w'^s$ if $s$ is the discrete log to base $g_y$, or $\pi_Y = w''^s$ if $s$ is the discrete log to base $g_x$. Hence for any $X$ and $\pi_X$, $Y$ can efficiently establish if $I$ is in the common language by comparing $\pi_X$ to $\pi_Y$, if they are equal for $w'$ and $w''$, then $DL2-T_\tau(X, Y, (\ell, w, g_x, g_y, q, p))$ is true, otherwise, it is false. It then follows that if $P.M_\tau(I)$ is true for TM $M$ (can be $X$, or $Y$) then $\exists \pi_M \forall \pi_Z$ such that $V.M_{T_\tau}(\pi_M, \pi_Z, I))$ is true.
• **τ-Hiding Proof Requirement for** $T_\tau(X, Y, I)$: Based on the intractability of the CDH assumption, there exists no DH solver that can output the DH key given only the key parts without knowledge of the discrete log of any of the key parts. It then follows that it should be hard to compute the CDH key for the key part $w$ and any key part $w'$ or $w''$ such that the discrete log of $w'$ to base $g_y$ is the same as the discrete log of $w''$ to base $g_x$. Hence, if $M$ can output a DH key constituting $\pi_M$ for all $\pi_Z$ on all key parts $w'$ or $w''$, then $M$ is a solver, but since a solver does not exists, then it must be that $M$ knows the discrete log of $w$ to base $g_x$ or to base $g_y$, i.e. that $P.M_\tau(\langle \ell, w, g_x, g_y, q, p \rangle)$ is true. It then follows that if the prover assertion does not hold for $M$, i.e., $\tau$ is false for $M$ and a given $I$, then it should not be able to establish if $\tau(Z, I)$ holds or not for any $Z$, since it can not efficiently derive the necessary $\pi_M$.

4.3 Common Languages for RSA

Dating back to 1801, when Gauss wrote in his Disquisitiones Arithmeticae, the problem of distinguishing prime numbers from composite numbers, and of resolving the latter into their prime factors, is known to be one of the most important and useful in arithmetic, and ever since then, the problem of integer factorization remains a corner stone of public key cryptography and is known to be in NP, particularly, NP-intermediate. One of the important problems that relies on the assumed hardness of integer factorization is the **RSA problem** [64].

Given an RSA public key $(m, e)$, where $m$, the modulus, is the product of two or more large primes, and $e$, the public exponent, is an odd integer $e \geq 3$ that is relatively prime
**RSA language and relation:**

- \( R_{RSA} = \{ (\langle W,m,e \rangle, \hat{M}^e = W \mod (m)) \} \).
- \( RSA = \{ (W,m,e) : (\exists \hat{M} : (\langle W,m,e \rangle, \hat{M} ) \in R_{RSA}) \} \).

<table>
<thead>
<tr>
<th>Table 4.4: NP Language of RSA.</th>
</tr>
</thead>
</table>

To \( \phi(m) \), the order of the multiplicative group defined by \( m \). Assume a ciphertext \( W = \hat{M}^e \mod n \), the RSA problem is to compute \( \hat{M} \).

To solve the RSA problem an adversary, who doesn’t know the private key \( e \), must nonetheless invert the RSA function. *The RSA Assumption is that the RSA Problem is hard to solve when the modulus \( m \) is sufficiently large and randomly generated, and when the plaintext \( \hat{M} \) (and hence the ciphertext \( W \)) is a random integer between 0 and \( m - 1 \).*

It follows from the RSA assumption that the RSA function is a trapdoor one-way function where the the private key \( d \) \((de = 1 \mod \phi(m))\) is the trapdoor. The randomness of the plaintext \( \hat{M} \) over the range \([0, m - 1]\) is important in the assumption. If \( \hat{M} \) is known to be from a small space, for instance, then an adversary can solve for \( \hat{M} \) by trying all possible values for \( \hat{M} \).

The RSA language and its underlying relation are given in Table 4.4 (the language members \( I \) is given by the vector enclosed in angular parenthesis).

For every vector \( I \) in \( RSA \), the value of \( \hat{M} \), constitutes a proof that \( I : \langle W,m,e \rangle \) is in \( RSA \). Unfortunately, testing for membership in an \( RSA \) language is considered trivial, just like in \( DL \), i.e., all \( I \) are trivially in \( RSA \). Traditional proof systems for \( RSA \) have focused on knowledge assertions rather than testing for membership. As mentioned earlier, this
involved defining knowledge extractors, making such systems vulnerable by definition to any attacks that can make use of these efficient knowledge extractor strategies. Using proofs of membership on common languages, it is possible to achieve a non-trivial proof system that avoid making direct knowledge assertions.

The idea is to construct proofs that use the knowledge of $X$ and $Y$ in similar ways to construct common values. Our RSA common language proposals are based on the abstract language $RSA$ and assume that $X$ and $Y$ each know some RSA exponent. We consider the case of identical knowledge, i.e., when both parties know the same exponent $e$, and the case of non identical knowledge, where $X$ knows an exponent $e$ and $Y$ knows its inverse, $d$ such that $(ed=1 \pmod{\pi})$.

It then follows that each of the parties can be used to construct an RSA solver based on its known exponent. For example, if $X$ knows an exponent $d$, then an RSA solver for exponent $e$ can be constructed based on $X$. This solver can produce $\hat{M}$ for any $\hat{W}$ such that $\hat{W} = \hat{M}^e$.

In keeping with the traditional language model, we choose to specify the prover strategy in terms of the RSA language defined in Table 4.4, i.e., we do not explicitly specify that an RSA exponent is known, instead we use knowledge of the relation $R_{RSA}$ to convey knowledge of the exponent.

Next, we need to define constraints on the RSA solvers for the common language relation. First, both solvers have to produce a common value. Second, we use universal quantification over all $W$ to prevent the use of accidental knowledge. Finally, we assume that the intractability assumption of RSA holds.

A note on identical versus non identical prover strategies: When $X$ and $Y$ know different RSA exponents, its important to find a basis for relating both. We choose to consider the
knowledge of an exponent or its inverse in specifying our knowledge assertion $\tau$, nonetheless, other assertions can be used as well. Hence, it is important to consider this difference in specifying the relation constraint. When the exponents known by $X$ and $Y$ are different, particularly, if $Y$ knows the inverse exponent, $d$, of the exponent $e$ known to $X$, then for a given $\hat{W}$ there exists $\hat{M}$ such that $\hat{W} = \hat{M}^e$ and the output of the RSA solver based on $Y$ on input $\hat{W}$ is the same as the output of the RSA solver based on $X$ on input $\hat{M}^d$.

4.3.1 RSA Common Language for Identical Prover Strategies

We start by identifying theorems $RSA1-\tau$, $RSA1-T_\tau$, the common $RSA1$ language $RSA1-L_{T_\tau}$, and the common relation $RSA1-R_{L_{T_\tau}(X,Y)}$ are listed in Table 4.4. The members of this language is of the form $I = \langle W, \hat{M}, m \rangle$. $RSA-S2$ denotes the RSA solver strategy that uses the RSA specific knowledge of a given party to solve the RSA problem for any RSA instances that uses $e$ as its exponent, such that $W = \hat{M}^e$.

Such solvers are made only possible through the knowledge of $X$ and $Y$ of the inverse exponent of $e$, otherwise, and with the assumed hardness assumption of RSA, such solvers do not exists or else they can be used to break the intractability of RSA. If $V.Y_{T_\tau}(\pi_Y, \pi_X, I))$ is true, then $Y$ can check that the output of the two solvers are equal. In this case a proof part $\pi_X$ is the $e$th root, $\hat{M}$ produced by the solver based on $X$ for any value $\hat{W}$. Similarly, the output of the solver based on $Y$ constitutes the proof part $\pi_Y$ which is the $e$th root, $\hat{M}$ for any value $\hat{W}$. The verification is an efficient equality check done on the two proof parts.

**Theorem 4.3** The common language $RSA2-L_{T_\tau}(X,Y)$ satisfies the requirements of the common language model.
RSA common language and common relation for parties $X$ and $Y$ designed for identical prover strategies:

- $RSA_{2-RLT}(X,Y) = \{(\langle W, \hat{M}, m \rangle, RSA-S2) : (\exists e \text{ such that } (W = \hat{M}^e) \land (\forall \hat{W} \exists \hat{M} (\hat{W} = \hat{M}^e) \land (RSA-S2(X, \hat{W}, e) = RSA-S2(Y, \hat{W}, e) = \hat{M}))))\}.$

- $RSA_{2-LT}(X,Y) = \{(W, \hat{M}, m) : (\exists e \exists RSA solver RSA-S2 : (((W, \hat{M}, m), RSA-S2) \in RSA_{2-RLT}(X,Y)) \land (RSA assumption holds)))\}.$

- $RSA_{2-\tau}(M, \langle W, \hat{M}, m \rangle) : (\exists e \text{ such that } (W = \hat{M}^e), \text{ and } (\forall \hat{W} M \text{ knows } \hat{M} \text{ such that } (\langle \hat{W}, m, e \rangle, \hat{M}) \in R_{RSA})).$

- $P.M_{\tau}(\langle W, \hat{M}, m \rangle) : M \text{ is an efficient strategy for proving } RSA_{2-\tau}(M, \langle W, \hat{M}, m \rangle).$

- $RSA_{2-T}(X, Y, \langle W, \hat{M}, m \rangle) : (RSA_{2-\tau}(X, \langle W, \hat{M}, m \rangle) \land RSA_{2-\tau}(Y, \langle W, \hat{M}, m \rangle)).$

Table 4.5: Specification of Theorems $RSA_{2-\tau}$ and $RSA_{2-T}$, and the Common Language $RSA_{2-LT}(X,Y)$
**Proof** We show that $RSA2-L_T(X,Y)$ satisfies the postulates of the model listed in Table 3.1.

- **Postulate 3.1:** Based on the definition of $\tau(M, \langle W, \hat{M}, m \rangle)$ in 4.5, for RSA exponent $e$ such that $W = \hat{M}^e$ and its inverse exponent $d$, such that $(de = 1 \pmod{\phi(m)})$, $M$ knows the $e$th of any value. Based on the hardness of finding these roots, and since in our model, $\tau$ has to be provable in polynomial time, then an efficient proof is for $M$ to output the inverse of the RSA exponent $e$, i.e., to output $d$ to show that it can compute the $e$th root of any value.

- **Postulate 3.2:** If $X$ is a prover for $\tau(X, I)$, then it can output $d$ such that $(de = 1 \pmod{\phi(m)})$, hence it can efficiently compute the $e$th root for any value $\hat{W}$. Hence for any $Y$ and $\pi_Y$, $X$ can efficiently establish if $I$ is in the common language by comparing $\pi_X$ to $\pi_Y$ for any random $\hat{W}$, if they are equal, then $RSA2-T_T(X,Y, \langle W, \hat{M}, m \rangle)$ is true, otherwise, it is false. Similarly, if $Y$ is a prover for $\tau(Y, I)$, then it can output $d$ such that $(de = 1 \pmod{\phi(m)})$, hence it can efficiently compute the $e$th root for any value $\hat{W}$. Hence for any $X$ and $\pi_X$, $Y$ can efficiently establish if $I$ is in the common language by comparing $\pi_X$ to $\pi_Y$ for any random $\hat{W}$, if they are equal, then $RSA2-T_T(X,Y, \langle W, \hat{M}, m \rangle)$ is true, otherwise, it is false. It then follows that if $P.M_T(I)$ is true for TM $M$ (can be $X$, or $Y$) then $\exists \pi_M \forall \pi_Z$ such that $V.M_T(\pi_M, \pi_Z, I))$ is true.

- **$\tau$-Hiding Proof Requirement for $T_T(X, Y, I)$:** Based on the intractability of the RSA assumption, there exists no RSA solver that can output the $e$th root without knowledge of the inverse of the RSA exponent $e$. It then follows that it should be
hard to compute the \textit{et}h root for all values \( \hat{W} \) based on the hardness RSA. Hence, if \( M \) can output the \( \text{et} \)h, \( \hat{M} \) constituting \( \pi_M \) for all \( \pi_Z \) for all values \( \hat{W} \), then \( M \) is a solver, but since a solver does not exists, then it must be that \( M \) knows the inverse of \( \text{et} \), i.e., that \( P.M\tau(\langle w, g, q, p \rangle) \) is true. It then follows that if the prover assertion does not hold for \( M \), i.e., \( \tau \) is false for \( M \) and a given \( I \), then it should not be able to establish if \( \tau(Z, I) \) holds or not for any \( Z \), since it can not efficiently derive the necessary \( \pi_M \).

\[ \blacksquare \]

### 4.3.2 RSA Common Language for non Identical Prover Strategies

Next, we consider a common language that allows for non identical prover strategies. Theorems \( \text{RSA2-} \tau \), \( \text{RSA2-} \hat{T} \), the common RSA2 language \( \text{RSA1-} \hat{L}, \), and the common relation \( \text{RSA2-} R_{\hat{L}(x,y)} \) are listed in Table 4.6. The members of this language is of the form \( I=\langle W, \hat{M}, m \rangle \). \( \text{RSA-S1} \) denotes the RSA solver strategy that uses the RSA specific knowledge of a given party to solve the RSA problem for any RSA instances that uses \( e \) or \( d \) as its exponent, such that \( W = \hat{M}^e \) and \( ed = 1 (mod \ \phi(m)) \).

Such solvers are made only possible through the knowledge of \( X \) and \( Y \) of the inverse exponent of \( e \), i.e., \( d \), or of the inverse exponent of \( d \), i.e., \( e \). Otherwise, and with the assumed hardness assumption of RSA, such solvers do not exists or else they can be used to break the intractability of RSA. If \( V.Y\tau(\langle \pi_Y, \pi_X, I \rangle) \) is true, then \( Y \) can check that the output of the two solvers are equal, possibly on different values of they know different exponents. In this case a proof part \( \pi_X \) is the \( e \)th or the \( d \)th root, \( \hat{M} \) produced by the solver based on \( X \) for any value \( \hat{W} \). Similarly, the output of the solver based on \( Y \) constitutes the proof part \( \pi_Y \) which is the \( e \)th or the \( d \)th root, \( \hat{M} \) for any value \( \hat{W} \). In the case where both \( X \) and \( Y \) know the same exponent, the solvers can produce the same output given the
RSA common language and common relation for parties X and Y designed for possibly non identical prover strategies:

- \( RSA_{1-\hat{L}T\tau}(X,Y) = \{ \langle W, \hat{M}, m \rangle : (\exists e_1, e_2 \in RSA\text{-}S1 \ (\forall \hat{W} \exists \hat{M} \text{ such that } \langle \hat{W}, \hat{M}, e_1 \rangle = RSA\text{-}S1(X, \hat{W}, e_1) = RSA\text{-}S1(Y, \hat{W}, e_2) = \hat{M}) \}) \}. \)

- \( RSA_{1-\hat{L}T\tau}(X,Y) = \{ \langle W, \hat{M}, m \rangle : (\exists e_1, e_2 \in RSA\text{-}S1 : (\langle W, \hat{M}, m \rangle, RSA\text{-}S1) \in RSA_{1-\hat{L}T\tau}(X,Y) \}) \}. \)

- \( RSA_{1-\hat{T}T\tau}(X,Y) = \{ \langle W, \hat{M}, m \rangle : (\exists e \in RSA \text{ such that } ((W = \hat{M}^e) \land (\forall \hat{W} \exists \hat{M} \text{ such that } \langle \hat{W}, \hat{M}, e \rangle, \hat{M}) \in RSA) \} \). \)

- \( P.M_{\tau}(\langle W, \hat{M}, m \rangle) : M \text{ is an efficient strategy for proving } RSA_{1-\hat{T}T\tau}(X,Y) \).

- \( RSA_{1-\hat{T}T\tau}(X,Y) = \{ (\exists e \in RSA \text{ such that } ((W = \hat{M}^e) \land (\forall \hat{W} \exists \hat{M} \text{ such that } \langle \hat{W}, \hat{M}, e \rangle, \hat{M}) \in RSA) \} \). \)

Table 4.6: Specification of Theorems \( RSA_{1-\tau} \) and \( RSA_{1-\hat{T}T\tau} \), and the Common language \( RSA_{1-\hat{L}T\tau}(X,Y) \).

same input \( \hat{W} \), whereas, when their exponents are different, then the output of one of the solvers for RSA exponent \( e \) on input \( \hat{W} \) will be equal to the output of the other solver for RSA exponent \( d \) on input \( \hat{M}^d \), and both outputs will be equal to \( \hat{M} \). The verification is an efficient equality check done on the two proof parts.

**Theorem 4.4** The common language \( RSA_{1-\hat{L}T\tau}(X,Y) \) satisfies the requirements of the common language model.
Proof  We show that $RSA_{1-\hat{T}_{t}(X,Y)}$ satisfies the postulates of the model listed in Table 3.1.

- **Postulate 3.1:** Based on the definition of $\tau(M, \langle \hat{W}, \hat{M}, m \rangle)$ in 4.6, for RSA exponent $e$ such that $W = \hat{M}^e$ and its inverse exponent $d$, such that $(de = 1 \pmod{\phi(m)})$, $M$ knows the $e$th or the $d$th root of any value. Based on the hardness of finding the these roots, and since in our model, $\tau$ has to be provable in polynomial time, then an efficient proof of is for $M$ to output the inverse of the RSA exponent $d$ to show that it can compute the $e$th root of any value, and to output the RSA exponent $e$ to show that it can compute the $d$th root of any value. This can be done in polynomial time if $M$ knows these inverses.

- **Postulate 3.2:** If $X$ is a prover for $\tau(X, I)$, then it can output the inverse of a given RSA exponent, hence it can efficiently compute the $e$th or the $d$th root for any value $\hat{W}$. Hence for any $Y$ and $\pi_Y$, $X$ can efficiently establish if $I$ is in the common language by comparing $\pi_X$ to $\pi_Y$ for any random $\hat{W}$ and $\hat{M}$ such that $\hat{W} = \hat{M}^e$, if they are equal, then $RSA_{1-\hat{T}_{t}(X, Y, \langle \hat{W}, \hat{M}, m \rangle)}$ is true, otherwise, it is false. The same argument applies for $Y$. It then follows that if $P.M_t(I)$ is true for TM $M$ (can be $X$, or $Y$) then $\exists \pi_M \forall \pi_Z$ such that $V.M_{T_t}(\pi_M, \pi_Z, I))$ is true.

- **$\tau$-Hiding Proof Requirement for $T_{t}(X, Y, I)$:** Based on the intractability of the RSA assumption, there exists no RSA solver that can output the $e$th or the $d$th root without knowledge of the corresponding inverse. It then follows that it should be hard to compute the root for all values $\hat{W}$ based on the hardness RSA. Hence, if $M$
can outputs the root then $M$ is a solver, but since a solver does not exists, then it must be that $M$ knows the inverse of the RSA exponent, i.e., that $P.M\tau(\langle W, \hat{M}, m \rangle)$ is true. It then follows that if the prover assertion does not hold for $M$, i.e., $\tau$ is false for $M$ and a given $I$, then it should not be able to establish if $\tau(Z, I)$ holds or not for any $Z$, since it can not efficiently derive the necessary $\pi_M$.

\[\Box\]

4.4 Common Languages for Hamiltonian Graphs

A Hamiltonian cycle in a graph $G(V, E)$ is a closed path that visits every vertex in the vertex set $V$ of the graph $G$ exactly once, ending up at the vertex from where it started. Hamiltonian graphs are named after the nineteenth-century Irish mathematician Sir William Rowan Hamilton (1805-1865). A graph which contains some Hamiltonian cycle is called Hamiltonian; otherwise, the graph is not considered a Hamiltonian graph. Not all graphs are Hamiltonian, e.g., every bi-partite graph with odd number of vertices is not Hamiltonian.

The problem of finding a Hamiltonian cycle in a undirected graph has been studied for many years and proven to be in NP complete via a so called Karp reduction. The language of Hamiltonian Graphs and its underlying relation are given in Table 4.7 (the language members $I$ is given by the vector enclosed in enclosed in angular parenthesis). Using proofs of membership on common languages, it is possible to achieve a proof system that avoid making direct assertions on the knowledge of a Hamiltonian cycle of a given graph. We propose common language designs based on the abstract language $HML$.

We consider finding a common computation that both $X$ and $Y$ can do using their knowledge of a Hamiltonian cycle of Graph $G$. Particularly, we consider the problem
Graph Hamiltonian language and relation:

- \( R_{HML} = \{(G, C) : C \text{ is a Hamiltonian Cycle of } G \} \)
- \( HML = \{G : \exists C : (G, C) \in R_{HML}\} \)

Table 4.7: NP Language of Hamiltonian Graphs.

of finding the isomorphism between two graphs \( G_1 \) and \( G_2 \). An isomorphism is a bijection \((f : \hat{V}_1 \to \hat{V}_2(G_2))\) between the vertex set \( \hat{V}_1 \) of \( G_1 \) and the vertex set \( \hat{V}_2 \) of \( G_2 \), such that any two vertices \( u \) and \( v \) of \( G_1 \) are adjacent in \( G_1 \) if and only if \( f(u) \) and \( f(v) \) are adjacent in \( G_2 \). If \( G_1 \) and \( G_2 \) are non isomorphic, then such an edge preserving bijection does not exists. Finding the isomorphism between two graphs is known to be a hard problem, i.e., there is no known efficient solver for this problem particularly for certain graphs and graph sizes (in terms of the number of edges and vertices).

Consider the case when a Hamiltonian cycle \( C_1 \) of graph \( G_1 \) is known, as well as its mapping, cycle \( C_2 \), in graph \( G_2 \). Then it can be easy to derive the isomorphism between \( G_1 \) and \( G_2 \) using \( C_1 \) and \( C_2 \) because the hardness of finding the isomorphism is related to the degree of the vertices. Hence, finding the isomorphism between two cycles is easy.

Hence, we consider finding the isomorphism as the common computation that each of \( \overline{X} \) and \( \overline{Y} \) can do using their own cycles, particularly instances of the isomorphism problem in which graph \( G \), for which \( \overline{X} \) and \( \overline{Y} \) has knowledge, is one of the two graphs. The prover strategies at \( \overline{X} \) and \( \overline{Y} \) can be used to construct efficient graph isomorphism solvers (GIS). These solvers have to produce as a common output, the isomorphism between graph \( G \) and any other graph \( G' \).
A note on non identical prover strategies: For non identical prover strategies at \( \Xi \) and \( \Psi \), we consider the case when \( \Xi \) knows a cycle \( C_\Xi \) of \( G_\Xi \) and \( \Psi \) knows another cycle \( C_\Psi \) of possibly another graph \( G_\Psi \). But to relate the respective knowledges of \( \Xi \) and \( \Psi \), we specify that there exists a common graph \( G \), such that \( G_\Xi = \eta G, \quad G_\Psi = \eta^{-1} G, \quad \) and \( C_\Xi = \eta C_\Psi \) for some isomorphism \( \eta \). It then follows that \( C_\Xi \) and \( C_\Psi \) are also cycles of \( G \). This allows us to create a common base for the relation between \( \Xi \) and \( \Psi \), which is the graph \( G \). Nonetheless, the solvers can use the graphs \( G_\Xi \) or \( G_\Psi \) based on its known cycles to help in solving for the isomorphism.

4.4.1 Hamiltonian Graphs Common Language for Identical Prover Strategies

We start by identifying theorems \( HML1-\tau, \quad HML1-T_\tau \), the common \( HML \) language \( HML1-L_{T_\tau} \), and the common relation \( HML1-R_{L_{T_\tau}(X,Y)} \) are listed in Table 4.8. The members of this language is of the form \( I=G \). GIS denotes the Graph Isomorphism solver strategy that uses the specific knowledge of a Hamiltonian cycle for graph \( G \) of a given party to solve the Isomorphism problem for any problem instance \((G_1,G_2)\) such that \( G \) is one of graphs \( G_1, \quad G_2 \).

Such solvers are made only possible through the knowledge of \( X \) and \( Y \) on \( G \), otherwise, and with the assumed hardness assumption of finding the isomorphism between two graphs \( G_1, \quad G_2 \), such solvers do not exists or else they can be used to break the intractability of the isomorphism problem. If \( V.Y_{T_\tau}(\pi_Y, \pi_X, I) \) is true, then \( Y \) can check that the output of the two solvers are equal. In this case a proof part \( \pi_X \) is the isomorphism produced between graphs \( G \) and any other isomorphic graph \( \hat{G} \). Similarly, the output of the solver based on \( Y \) constitutes the proof part \( \pi_Y \) which is the isomorphism between graph \( G \) and any other
HML common language and common relation for parties \( X \) and \( Y \) designed for identical prover strategies:

- \( HML1-R_{T_\tau(X,Y)} = \{ (G, GIS) : (\forall \hat{G} \text{ such that } (\exists C ((\hat{G}, C) \in R_{HML}) \land (GIS(X, \hat{G}, C, G) = GIS(Y, \hat{G}, C, G) = \eta) \land (\hat{G} = \eta G, \text{ i.e., } \eta \text{ is an isomorphism of } \hat{G} \text{ and } G)))) \} \).

- \( HML1-\hat{L}_{T_\tau(X,Y)} = \{ G : (\exists \text{ Graph Isomorphism solver GIS: } ((G, GIS) \in HML1-R_{L_{T_\tau(X,Y)}})) \} \)

- \( HML1-\tau(M, G) : M \) knows Hamiltonian cycle \( C \) such that \( (G, C) \in R_{HML} \).

- \( P.M_{\tau}(G) : M \) is an efficient strategy for proving \( HML1-\tau(M, G) \).

- \( HML1-T_{\tau}(X, Y, G) : HML1-\tau(X, G) \land HML1-\tau(Y, G) \).

Table 4.8: Specification of Theorems \( HML1-\tau \) and \( HML1-T_\tau \), and the Common Language \( HML1-\hat{L}_{T_\tau(X,Y)} \).

The verification is an efficient equality check done on the two proof parts.

**Theorem 4.5** The common language \( HML1-\hat{L}_{T_\tau(X,Y)} \) satisfies the requirements of the common language model.

**Proof** We show that \( HML1-\hat{L}_{T_\tau(X,Y)} \) satisfies the postulates of the model listed in Table 3.1.

- **Postulate 3.1:** Based on the definition of \( \tau(M, I) \) in 4.8, \( M \) knows a Hamiltonian cycle \( C \) of \( G \), hence, \( M \) can prove that it has knowledge by simply outputting \( C \), a
Hamiltonian cycle of $G$, which can be done in polynomial time, since $M$ knows $C$.

- **Postulate 3.2:** If $X$ is a prover for $\tau(X, I)$, then it can output $C$ such that $C$ is a Hamiltonian cycle of $G$, hence it can efficiently compute the isomorphism between any two graphs, as long as $G$ is one of them, by deriving the isomorphism between the cycles of each. Hence for any $Y$ and $\pi_Y$, $X$ can efficiently establish if $I$ is in the common language by comparing $\pi_X$ to $\pi_Y$ for the isomorphism between $G$ and any random graph $\hat{G}$, if they are equal, then $HML1-T_\tau(X, Y, G)$ is true, otherwise, it is false. Similarly, if $Y$ is a prover for $\tau(Y, I)$, then it can output $C$ such that $C$ is a Hamiltonian cycle of $G$, hence it can efficiently compute the isomorphism between any two graphs, as long as $G$ is one of them, by deriving the isomorphism between the cycles of each. Hence for any $X$ and $\pi_X$, $Y$ can efficiently establish if $I$ is in the common language by comparing $\pi_Y$ to $\pi_X$ for the isomorphism between $G$ and any random graph $\hat{G}$, if they are equal, then $HML1-T_\tau(X, Y, G)$ is true, otherwise, it is false. It then follows that if $P.M_\tau(I)$ is true for TM $M$ (can be $X$, or $Y$) then $\exists \pi_M \forall \pi_Z$ such that $V.M_{T_\tau}(\pi_M, \pi_Z, I))$ is true.

- **$\tau$-Hiding Proof Requirement for $T_\tau(X, Y, I)$:** Based on the intractability of the isomorphism between two graphs, there exists no graph isomorphism solver for two graphs $(G, \hat{G})$ for any $\hat{G}$, that can output the isomorphism given only a cycle of graph $\hat{G}$. Nonetheless, if a cycle in $G$ is known, then it is possible to derive the isomorphism. It then follows that if $M$ can output the isomorphism constituting $\pi_M$ for all $\pi_Z$ for graph $G$ and any graph $\hat{G}$, then $M$ is a solver, but since a solver does not exists, then it must be that $M$ knows a cycle of $G$, i.e.that $P.M_\tau(G)$ is true. It then follows
that if the prover assertion does not hold for \( M \), i.e., \( \tau \) is false for \( M \) and a given \( I \), then it should not be able to establish if \( \tau(Z,I) \) holds or not for any \( Z \), since it can not efficiently derive the necessary \( \pi_M \).

\[ \blacksquare \]

### 4.4.2 Hamiltonian Graphs Common Language for non Identical Prover Strategies

Table 4.9 lists assertions \( HML2-\tau \), \( HML2-T_\tau \), the common \( HML \) language \( HML2-L_T \), and the common relation \( HML2-R_{L_T(X,Y)} \) that allows for non identical prover strategies at \( X \) and \( Y \). The members of this language is of the form \( I = \langle G, G_x, G_y \rangle \). GIS denotes the Graph Isomorphism solver strategy that uses the specific knowledge of a Hamiltonian cycle for graph \( G \) of a given party to solve the Isomorphism problem for any problem instance \((G_1, G_2)\) such that \( G \) is one of graphs \( G_1, G_2 \). The idea is to enable \( X \) that knows a cycle \( C_x = C \) of \( G \) to recognize that \( Y \) also knows a possibly non identical cycle \( C_y = \eta C \) of \( G \), where \( \eta \) can be an identity permutation or not. Nonetheless, deriving the isomorphism between graphs \( G \) and any other graph \( \hat{G} \) for which a cycle is known becomes harder as \( X \) will always derive an isomorphism shifted by \( \eta \) from the isomorphism derived by \( Y \), i.e., the output of the isomorphism solver (GIS) based on \( X \) will not be identical to the output of the GIS based on \( Y \) for graphs \( G \) and \( \hat{G} \) for which a cycle \( \hat{C} \) is known, particularly for the case when \( \eta \) is not the identity permutation.

To find a common output for the two solvers, we consider only cycles that are shifted by a valid isomorphism \( \eta \), i.e., an edge preserving permutation, hence, \( \eta \) can be used to derive graphs \( G_x \) and \( G_y \) such that \( G_x = \eta G \) and \( G_y = \eta^{-1} G \). Cycle \( C_x \) maps to cycle \( C_y \) in \( G_y \), and cycle \( C_y \) maps to cycle \( C_x \) in graph \( G_x \). It then follows that for any graph \( \hat{G} \) such
that \( \hat{C} \) is a cycle of that graph, the solver based on \( X \) can map \( \hat{C} \) to \( C_x \) of \( G_x \), and the solver based on \( Y \) can map \( \hat{C} \) to \( C_y \) for graph \( G \). Assuming \( \hat{C} \) is the mapping of cycle \( C_y \) in \( \hat{G} \), then both solvers will output the same isomorphism. Similarly if the the solver based on \( Y \) can map \( \hat{C} \) to \( C_y \) of \( G_y \), and the solver based on \( X \) can map \( \hat{C} \) to \( C_x \) for graph \( G \). Assuming \( \hat{C} \) is the mapping of cycle \( C_x \) in \( \hat{G} \), then both solvers will output the same isomorphism.

Such solvers are made only possible through the knowledge of \( X \) and \( Y \) on \( G \), otherwise, and with the assumed hardness assumption of finding the isomorphism between two graphs \( G_1 \) and \( G_2 \), such solvers do not exists or else they can be used to break the intractability of the isomorphism problem. If \( V.YT_x(\pi_Y, \pi_X, I)) \) is true, then \( Y \) can check that the output of the two solvers are equal. In this case a proof part \( \pi_X \) is the isomorphism produced between graph \( G \) or graph \( G_x \) and any other isomorphic graph \( \hat{G} \). Similarly, the output of the solver based on \( Y \) constitutes the proof part \( \pi_Y \) which is the isomorphism between graph \( G \) or graph \( G_y \) and any other isomorphic graph \( \hat{G} \). The verification is an efficient equality check done on the two proof parts as discussed earlier.

**Theorem 4.6** The common language \( HML2-L_{T_x}(X,Y) \) satisfies the requirements of the common language model.

**Proof** We show that \( HML2-L_{T_x}(X,Y) \) satisfies the postulates of the model listed in Table 3.1.

- **Postulate 3.1:** Based on the definition of \( \tau(M, I) \) in 4.9, \( M \) knows a Hamiltonian cycle \( C \) or \( \eta C \) of \( G \) such that \( G_x = \eta G \), and \( G_y = \eta^{-1} G \). Hence, \( M \) can prove that it has knowledge by simply outputting \( C \) or \( \eta C \circ G \), which can be done in polynomial time.
HML common language and relation for parties X and Y designed for possibly non-identical prover strategies:

- **HML2-RL\(L_{\tau(\langle X, Y \rangle)})\)=\{\langle G, G_x, G_y \rangle, \text{GIS}: (\exists G_x = \eta G \text{ and } G_y = \eta^{-1} G, \text{ then } (\forall \hat{G} \text{ isomorphic to } G, \exists \hat{C} \text{ such that } (((\hat{G}, \hat{C}) \in R_{HML}) \land (\text{GIS}(X, \hat{G}, \hat{C}, G) = \eta)) \lor (\text{GIS}(X, \hat{G}, \hat{C}, G_x) = \text{GIS}(Y, \hat{G}, \hat{C}, G_y) = \eta)) \land (\hat{G} = \eta G, \text{ i.e., } \eta \text{ is an isomorphism of } \hat{G} \text{ and } G)))\}\}.

- **HML2-\(\hat{L}_{\tau(\langle X, Y \rangle)})\)=\{\langle G, G_x, G_y \rangle : (\exists \text{ Graph Isomorphism solver GIS: } (((\langle G, G_x, G_y \rangle, \text{GIS}) \in HML2-RL_{\tau(\langle X, Y \rangle)})})\}\}.

- **HML2-\(\tau(M, \langle G, G_x, G_y \rangle)\): (\exists \eta \exists C \text{ such that } ((C \text{ and } \eta C \text{ are the Hamiltonian cycles in } G, \text{ i.e., } ((G, C) \in R_{HML} \land (G, \eta C) \in R_{HML}) \land (G_x = \eta G) \land (G_y = \eta^{-1} G)). M \text{ knows } \hat{C} \text{ such that } ((\hat{C} = C) \lor (\hat{C} = \eta C)) ) .

- **P.M\(\tau(\langle G, G_x, G_y \rangle): M \text{ is an efficient strategy for proving } HML2-\(\tau(M, \langle G, G_x, G_y \rangle)\).}

- **HML2-T\(\tau(X, Y, \langle G, G_x, G_y \rangle)\): HML2-\(\tau(X, \langle G, G_x, G_y \rangle) \land HML2-\(\tau(Y, \langle G, G_x, G_y \rangle)\).}

Table 4.9: Specification of Theorems \(HML2-\tau\) and \(HML2-T\(\tau\), and the Common Language \(HML2-\hat{L}_{\tau(\langle X, Y \rangle)}\).
• **Postulate 3.2:** If $X$ is a prover for $\tau(X, I)$, then it can output $C$ such that $C$ is a Hamiltonian cycle of $G$ and $G_x$, hence it can efficiently compute the isomorphism between any two graphs, as long as $G$ or $G_x$ is one of them, by deriving the isomorphism between the cycles of each. Hence for any $Y$ and $\pi_Y$, $X$ can efficiently establish if $I$ is in the common language by comparing $\pi_X$ to $\pi_Y$ for the isomorphism between $G$ or $G_x$ and any random graph $\hat{G}$, if they are equal, then $HML2-T_\tau(X, Y, G)$ is true, otherwise, it is false. The same argument holds true for $Y$. It then follows that if $PM_\tau(I)$ is true for TM $M$ (can be $X$, or $Y$) then $\exists \pi_M \forall \pi_Z$ such that $VM_{T_\tau}(\pi_M, \pi_Z, I))$ is true.

• **$\tau$-Hiding Proof Requirement for $T_\tau(X, Y, I)$:** Based on the intractability of the isomorphism between two graphs, there exists no graph isomorphism solver for two graphs $G$ or $G_M$ and any $\hat{G}$, that can output the isomorphism given only a cycle of graph $\hat{G}$. Nonetheless, if a cycle in $G$ and $G_M$ is known, then it is possible to derive the isomorphism. It then follows that if $M$ can output the isomorphism constituting $\pi_M$ for all $\pi_Z$ for graph $G$, or $G_M$ and any graph $\hat{G}$, then $M$ is a solver, but since a solver does not exists, then it must be that $M$ knows a cycle of $G$, i.e., $PM_\tau(G)$ is true. It then follows that if the prover assertion does not hold for $M$, i.e., $\tau$ is false for $M$ and a given $I$, then it should not be able to establish if $\tau(Z, I)$ holds or not for any $Z$, since it can not efficiently derive the necessary $\pi_M$. 

\[\blacksquare\]
4.5 Common Languages for Non-Isomorphic Graphs

Given two graphs, the problem of determining if these two graphs are non isomorphic is considered a hard problem and is known to be in the complexity class co-NP. A general challenge for problems in co-NP is the lack of a witness or proof. For example consider the NP language of Hamiltonian graphs, a cycle is a proof that a graph is in the language, whereas to prove that two graphs are non isomorphic, it is necessary to prove that there is no isomorphism, and to establish this truth, there is no explicit witness that can be used. The language of Non Isomorphic Graphs is given in Table 4.10 (the language members I is given by the vector enclosed in enclosed in angular parenthesis). The relation $R_{ISO}$ is the relation for the NP language of isomorphic graphs. If two graph $G_1$ and $G_2$ are non-isomorphic, then there does not exists an isomorphism $\pi$ such that the : $(\langle G_1, G_2 \rangle, \pi) \in R_{ISO}$.

Such solvers are made only possible through the common knowledge of $X$ and $Y$ on $G_1$, otherwise, and with the assumed hardness assumption of finding the isomorphism between two graphs $G_1$, and $\hat{G}$, such solvers do not exists or else they can be used to break the intractability of the isomorphism problem. If $V.Y_{T_2}(\pi_Y, \pi_X, I)$) is true, then $Y$ can check that the output of the two solvers are equal. In this case a proof part $\pi_X$ is the isomorphism produced between graphs $G_1$ and any other isomorphic graph $\hat{G}$. The same argument holds true for $Y$. The verification is an efficient equality check done on the two proof parts.

Proofs on common languages can be used for languages in CO-NP by assuming that the parties $X$ and $Y$ have relevant knowledge on the problem input. For graphs non-isomorphism, assume it is required to prove that $I = \langle G_1, G_2, \hat{V} \rangle$ is in a common non-isomorphic language of $X$ and $Y$. If $G_1$ and $G_2$ are non-isomorphic, then if $X$ and $Y$ have knowledge of this non isomorphism, then given a graph $\hat{G}$ which is isomorphic to either $G_1$ or $G_2$, both $X$ and $Y$ should be able to tell if $\hat{G}$ is isomorphic to $G_1$ or to $G_2$. In order to
Non-Isomorphic Graphs language and relation:

- $R_{ISO} = \{ ((G_1, G_2), \pi) : G_1 = \pi G_2 \}.$
- $NOI = \{ (G_1, G_2) : (\neg \exists \pi : ((G_1, G_2), \pi) \in R_{ISO}) \}.$

Table 4.10: co-NP Language of non-Isomorphic Graphs.

achieve this, we let $X$ and $Y$ have knowledge of a Hamiltonian cycle in one of the graphs, say $G_1$, then this cycle can be used by both to determine if there is a valid permutation constituting an isomorphism between $G_1$ and $\hat{G}$. If the permutation is valid, then $\hat{G}$ is isomorphic to $G_1$, otherwise, it is isomorphic to $G_2$. We start by identifying theorems $NOI-\tau$, $NOI-T_\tau$, the common $NOI$ language $NOI-\hat{L}_{T_\tau}$, and the common relation $NOI-R_{L_{T_\tau}(X,Y)}$ are listed in Table 4.11.

Theorem 4.7  The common language $NOI-\hat{L}_{T_\tau}(X,Y)$ satisfies the requirements of the common language model.

Proof  We show that $NOI-\hat{L}_{T_\tau}(X,Y)$ satisfies the postulates of the model listed in Table 3.1.

- Postulate 3.1: Based on the definition of $\tau(M,I)$ in 4.11, $M$ knows that the two graphs $G_1$ and $G_2$ are non isomorphic, and it knows a Hamiltonian cycle $C$ of $G_1$, hence, $M$ can prove that $G_1$ and $G_2$ are non isomorphic by determining if a given graph ($\hat{G}$), and a cycle ($\hat{C}$) of this graph) is isomorphic to $G_1$ or isomorphic to $G_2$.  

87
NOI common language and relation for parties $X$ and $Y$ designed for identical prover strategies

- $NOI-R_{L\tau(X,Y)} = \{ (\langle G_1, G_2 \rangle, GIS) : (\forall \hat{G} \exists \eta \text{ such that } ((\langle \hat{G}, G_1 \rangle) \in NOI) \land ((\langle \hat{G}, G_2 \rangle) \in NOI) \land (GIS(X, \hat{G}, G_1) = GIS(Y, \hat{G}, G_1) = \eta) \land (\eta \text{ is a valid isomorphism if and only if } (\langle \hat{G}, G_1 \rangle, \eta) \in R_{ISO})) \} \}$

- $NOI-\hat{L}_{T\tau}(X,Y) = \{ \langle G_1, G_2 \rangle : (\exists \text{ Graph Isomorphism solver GIS: } ((\langle G_1, G_2 \rangle, GIS) \in HML1-\hat{R}_{L\tau(X,Y)}) \} \}$

- $NOI-\tau(M, \langle G_1, G_2 \rangle) : M \text{ knows that there does not exists } \pi \text{ such that } (\langle G_1, G_2 \rangle, \pi) \in R_{ISO} \text{ and } HML1-\tau(\langle G \rangle) \text{ is true.}$

- $P.M_{\tau}(\langle G_1, G_2 \rangle) : M \text{ is an efficient strategy for proving } NOI-\tau(M, \langle G_1, G_2 \rangle).$

- $NOI-T_{\tau}(X, Y, \langle G_1, G_2 \rangle) : NOI-\tau(X, \langle G_1, G_2 \rangle) \land NOI-\tau(Y, \langle G_1, G_2 \rangle)$

Table 4.11: Specification of Theorems $NOI-\tau$ and $NOI-T_{\tau}$, and the Common Language $NOI-\hat{L}_{T\tau}(X,Y)$. 

88
This can be done in polynomial time using the cycle \( C \) of \( G_1 \), by deriving the permutation between \( C \) and \( \hat{C} \) and then checking if this permutation \( \eta \) is a valid isomorphism (i.e., if it is edge preserving such that \( G = \eta G_1 \)). If it is a valid isomorphism, then \( \hat{G} \) is isomorphic to \( G_1 \), otherwise, it is isomorphic to \( G_2 \).

- **Postulate 3.2:** If \( X \) is a prover for \( \tau(X, I) \), then it can output \( C \) such that \( C \) is a Hamiltonian cycle of \( G_1 \), hence it can efficiently compute the isomorphism between any two graphs, as long as \( G_1 \) is one of them, by deriving the isomorphism between the cycles of each. On the other hand, if \( G_1 \) is not isomorphic to the given graph \( \hat{G} \), then the permutation derived through the cycle will not result in a valid permutation. Hence for any \( Y \) and \( \pi_Y \), \( X \) can efficiently establish if \( I \) is in the common language by comparing \( \pi_X \) to \( \pi_Y \) for the isomorphism between \( G \) and any random graph \( \hat{G} \), if they are equal, then \( NOI-T_\tau(X, Y, \hat{G}) \) is true, otherwise, it is false. The same argument applies to \( Y \). It then follows that if \( P.M_\tau(I) \) is true for TM \( M \) (can be \( X \), or \( Y \)) then \( \exists \pi_M \forall \pi_Z \) such that \( V.M_\tau(\pi_M, \pi_Z, I) \) is true.

- **\( \tau \)-Hiding Proof Requirement for \( T_\tau(X, Y, I) \):** Based on the intractability of the isomorphism between two graphs, there exists no graph isomorphism solver for two graphs \((G_1, \hat{G})\) for any \( \hat{G} \), that can output the isomorphism given only a cycle of graph \( \hat{G} \). Nonetheless, if a cycle in \( G \) is known, then it is possible to derive the isomorphism. It then follows that if \( M \) can output the isomorphism constituting \( \pi_M \) for all \( \pi_Z \) for graph \( G_1 \) and any graph \( \hat{G} \), then \( M \) is a solver, but since a solver does not exists, then it must be that \( M \) knows a cycle of \( G_1 \), i.e.that \( P.M_\tau(G) \) is true. It
then follows that if the prover assertion does not hold for $M$, i.e., $\tau$ is false for $M$ and a given $I$, then it should not be able to establish if $\tau(Z,I)$ holds or not for any $Z$, since it can not efficiently derive the necessary $\pi_M$.

\[\square\]

4.6 Security Analysis and Adversary Model

In this section, we define the adversary model and security game used in our analysis of the security of protocols for proofs on common languages. We adopt the computational model approach. In this model, messages of the protocols are assumed to be bit-strings from some distribution, and the adversary is assumed to be an arbitrary algorithm. The cryptography is assumed to be an algorithm (or tuple of algorithms) which satisfy some asymptotic property even in the presence of an arbitrary adversary.

To prove the correctness of a protocol in this model, we construct reductions from the protocol to the underlying hard problem. That is, we show that if there exists an adversary $A$ with a significant chance of successfully attacking the protocol, then it can be used to construct an adversary $\hat{A}$ with a significant chance of breaking the underlying hard problem. Hence, one can conclude that if the intractability assumption holds for the hard problem, then the protocol must be secure.

We follow the traditional assumption that cryptographic adversaries are generally probabilistic and are required to work only for a non-negligible fraction of the inputs. It then suffices to take a single instance $j$ of the underlying hard problem and convert it to some instance $k$ for our problem provided that the instance $k$ look identical to an adversary $A$ in the security game. In other words, it must appear as though it came from a real distribution over all possible instances. We express the security of our protocols using the advantage
of an adversary, i.e., the probability that he succeeds over a naive algorithm in breaking the protocol. Our reduction uses an adversary $A$ that works with non-negligible advantage to construct an adversary $\hat{A}$ to break the underlying hard problem with nonnegligible advantage.

The computational model is known to be a strong model for analysis. The only assumption placed on the adversary is that it is efficient, i.e., it is executing in probabilistic polynomial time (PPT), which is a fairly weak assumption, giving the model a solid and meaningful grounding in complexity theory.

The proposed protocols are for identification, hence, we consider impersonation as one of the main adversary’s goals: it wins if it can interact with the receiver (in the role of a sender), and convince the latter to accept, i.e., establish that the assertion on a given input is indeed valid.

We consider two types of attacks, namely: passive and active attacks. Passive attacks correspond to eavesdropping, meaning the adversary is in possession of transcripts of conversations between the real sender and the receiver. Active attacks means that it gets to play the role of a receiver, interacting with the real sender and trying to extract information from it.
Chapter 5: Identical Prover Strategies for Common Languages

In this chapter we study protocols for proofs on the common languages defined for identical prover strategies in Chapter 4. We consider proofs on the common discrete log language $DL1-\hat{L}_T(X,Y)$ and the common graph Hamiltonian language $HML1-\hat{L}_T$. All the protocols constitute an interaction on a public common input $I$ that is not certified and can be proposed by any of the two parties $X$ or $Y$. The interaction on the common input constitutes a proof of membership of $I$ in the corresponding common language.

5.1 CDLL Protocol for Identification based on Proofs on a Common Discrete Log Language

In this section we present CDLL, a two party interactive protocol for identification based on membership in a Common Discrete Log Language (CDLL). The protocol is a proof of membership in the common language $DL1-\hat{L}_T(X,Y)$ specified in Table 4.2.

The protocol is based on the hardness of the Decision Diffie-Hellman problem (DDH). The common input $I = \langle \ell, g, q, p \rangle$ is defined in terms of $p$ and $q$, where $q$ is a prime factor of $p - 1$, i.e., $q | (p - 1)$. $q \geq 2^{140}$, $p \geq 2^{512}$, a generator $g$. Honest Parties $\overline{X}$ and $\overline{Y}$ knows the discrete log of $\ell$ to the base $g$. We assume that $I = \langle \ell, g, q, p \rangle$ is a public common input of both parties.
The CDLL protocol uses two messages. The first is sent from $Y$ to $X$ and contains a DH key part $B = g^b$ as well as a random $n$. The second is sent from $X$ to $Y$, and contains a DH key part $M = g^r \pmod{p}$, $R = ns_x + a \pmod{q}$, and a DH key $K$ derived using $n, r, a$ and $B$. $Y$ uses the verification expression $K = M^{b(R-ns_y)} \pmod{p}$ to check if $I$ is in $DL_1-\hat{L}_T(X,Y)$. This verification can only succeed if and only if for both $X$ and $Y$, $s_x = s_y = s$. The protocol is described formally in Table 5.1.

**Theorem 5.1**  
$CDLL(X, Y)$ is a proof system of membership in a common DL language based on the hardness of DDH.

**Proof**  
Assume negligible function $\varepsilon : N \to [0, 1] : \varepsilon(n) = 2^{-n}$. We prove Completeness, Prover and Verifier Soundness in that order.
Completeness: \( \bar{X} \) successfully responds to \( \bar{Y} \)'s challenge using its \( s_x \) to formulate \( R \). It can then easily compose the DH key \( K \) by using its private exponents \( a \) and \( r \) as well as \( Y \)'s key part \( B \) to produce \( K = B^{ar} \). Since \( \bar{Y} \) knows \( s_y = s_x \), it successfully verifies:

\[
M^{b(R-nsy)} \pmod{p} \\
= g^{rb(R-nsy)} \pmod{p} \\
= g^{br(R-nsy)} \pmod{p} \\
= B^{r(R-nsy)} \pmod{p} \\
= B^{r(a)} \pmod{p} \\
= K
\]

Hence, \( \bar{X} \) succeeds with probability 1 in convincing \( \bar{Y} \) that \( I \) is in the common language \( DL1-\hat{L}_T(X, \bar{Y}) \). This proves completeness with probability 1.

Prover Soundness: To establish prover soundness, assume dishonest party, \( \tilde{X} \), i.e., \( \tilde{X} \) does not know \( s_x \) such that \((g)^{s_x} = g^s = g^{sy} \). We prove that under these assumptions, \( \bar{Y} \) accepts the proof of \( \tilde{X} \) as valid with probability bounded by \( \epsilon(|I|) \).

\( \tilde{X} \) knows \( \ell \) but does not know \( s \), then it can cheat by guessing the correct value of \( b \) used in \( B \), and then choose an exponent \( r \), and send \( M = g^r \), as well as a randomly chosen value for \( R \), and \( K = B^{rR} I^{-nrb} \). Nonetheless, the probability of success of this strategy is only \( \epsilon(|I|) \).

The CDH assumption asserts that for all key parts \( R_1, R_2 \), and generator \( g \), it is hard to compute the Diffie-Hellman key \( \hat{K} \). We show that if there exists a probabilistic polynomial time algorithm \( \hat{A} \) that breaks prover soundness for CDLL, i.e., convinces \( \bar{Y} \) that \( I \) is in a common language with non negligible probability, then for all \( g \), and for all key part \( R_2 \) there exists key part \( R_1 \) such that a probabilistic polynomial time algorithm \( \hat{A} \) can output
the Diffie-Hellman key $\hat{K}$ using $A$ that neither knows the discrete log of $R_2 = \ell = g^s$, nor knows the discrete log of $R_1$, which contradicts the CDH intractability assumption.

$\hat{A}(g, R_2)$

- Select random $b$, and random $n$, and send $B = g^b$ along with $n$ to $A$.
- Receive $M$ and $K$.
- Compute $R_1 = M^b$
- Return Diffie Hellman key $R_1^{-R}K^{-1/n}$

If $A$ succeeds with non negligible probability in breaking $CDLL$, then $\hat{A}$ will be able to successfully find the DH key for the Diffie Hellman key parts $R_1$ and $R_2$ and generator $g$ for the randomly generated key part $R_1$ which is dependent on both $B$ and $M$, such that neither $\hat{A}$ nor $A$ knows the discrete log of $R_1$ or $R_2$. The construction $\hat{A}$ is an efficient one, and is guaranteed to succeed with non negligible probability if adversary $A$ succeeds in impersonating $\overline{X}$ that knows the discrete log of $\ell$ to base $g$ with non negligible probability. $\hat{A}$ derives the DH key of $R_1$ and $R_2$, by computing $R_1^{-R}K^{-1/n} = M^{(-bR)}M^{b(R-ns)} = R_1^{s}$, which is then returned as the DH key of $R_1$ and $R_2$.

Hence, we conclude that since there does not exists $\hat{A}$ that breaks the CDH assumption in polynomial time with non negligible probability, then there does not exist $A$ that breaks $CDLL$ soundness in polynomial time with non negligible probability.

**Verifier Soundness:** To prove verifier soundness, we shall consider all polynomial time strategies $D$ and black box interactions with $\overline{X}$ and $\tilde{X}$ or $\overline{Y}$ and $\tilde{Y}$. In the distinguishing experiment of $\overline{X}$, $D$ impersonates $\overline{Y}$ to $\overline{X}$, i.e., it can be considered as $\tilde{Y}$, since it does not have knowledge of the discrete log $s$. On the other hand, in the distinguishing experiment
involving $\widetilde{Y}$, $D$ impersonates $X$ to $\widetilde{Y}$, i.e., it can be considered as $\widetilde{X}$, since it does not have knowledge of the discrete log $s$.

**$D$ as $\widetilde{Y}$:** Assume any dishonest party, $\widetilde{Y}$, i.e., $\widetilde{Y}$ does not know $s_y$ such that $(g)^{s_y} = \ell$. Let $Pr^X_{\widetilde{Y}}(\tau(X,I))$ denotes the probability that $\widetilde{Y}$ outputs 1 indicating that $X$ can prove knowledge of $I$. To establish the first condition for verifier soundness, we prove that: $|Pr^X_{\widetilde{Y}}(\tau(X,I)) - Pr^X_{\widetilde{Y}}(\tau(\overline{X},I))| \leq \varepsilon(|I|)$, i.e., $\widetilde{Y}$ can not distinguish $\overline{X}$ from any other $X$. Note that the knowledge of a valid $s_x$, such that $(g)^{s_x} = (g)^{s_y}$ is what distinguishes $X$ from any other $X$.

We show that if there exists a probabilistic polynomial time algorithm $D$ that breaks verifier soundness for CDLL, i.e., $D$ can distinguish $\overline{X}$ from any other $X$ on a given $I$ ($|Pr^X_D(\tau(X,I)) - Pr^X_D(\tau(\overline{X},I))| > \varepsilon(|I|)$), then we construct a probabilistic polynomial time algorithm $\hat{A}$ that breaks the DDH intractability assumption. Recall that the DDH assumption asserts that for generators $g$, for all key parts $R_1, R_2$, it is hard to distinguish $(g, R_1, R_2, \hat{K})$ from $(g, R_1, R_2, K_r)$ such that $\hat{K}$ is the Diffie-Hellman key of the key parts $R_1$ and $R_2$ and $K_r$ is a random value. We show that if there exists a probabilistic polynomial time algorithm $D$ that breaks verifier soundness for CDLL, i.e., succeeds in distinguishing $\overline{X}$ from $X$ with non negligible probability, then for all $g$, and for all key part $R_2$ there exists key part $R_1, \hat{K}$ such that a probabilistic polynomial time algorithm $\hat{A}$ can decide if $\hat{K}$ is the Diffie-Hellman key for $R_1$ and $R_2$ or not. In our security game, $\hat{A}$ uses $D$ that neither knows the discrete log of $R_2 = \ell = g^s$, nor knows the discrete log of $R_1$. at any time of this game, $\hat{A}$ can ask for its challenge key $\hat{K}$ for $R_1$ and the fixed $R_2$. $\hat{A}$ is then given either the Diffie Hellman key of $R_1$ and $R_2$, i.e., $\hat{K} = \hat{K}$ or with any random value, i.e., $\hat{K} = K_r$. If $\hat{A}$ succeeded in correctly answering the security game challenge, then it is a solver for DDH, which contradicts the DDH intractability assumption.
\[ \hat{A}(g, R_2) \]

- Receive \( B \) and \( n \).
- Select random \( r \), compute \( R_1 = B^r \)
- Output \( R_1 \), and receive the challenge key \( \hat{K} \)
- send \( M = g^r \), random \( R \), and \( K = R_1^{R}\hat{K}^n \)
- Based on the output of \( D \), output 1, or 0.

If \( D \) succeeds with non negligible probability in breaking CDLL, then \( \hat{A} \) will be able to successfully determine if \( \hat{K} \) is the correct DH key of \( R_1 \) and \( R_2 \). The construction \( \hat{A} \) is an efficient one, and is guaranteed to succeed with non negligible probability if adversary \( A \) succeeds in distinguishing \( \tilde{X} \) from \( X \) with non negligible probability and if \( R_1 \) is the DH key of \( M \) and \( B \). \( \hat{A} \) sends random \( R \) and \( K = R_1^{R}\hat{K}^n = M^{bR}g^{rs} \) if \( \hat{K} \) is the correct DH key for \( R_1 \) and \( R_2 \), then the response \( M \) and \( K \) can be successfully verified, otherwise, it should fail.

Hence, we conclude that since there does not exists \( \hat{A} \) that breaks the DDH assumption in polynomial time with non negligible probability, then there does not exist \( D \) that breaks CDLL verifier soundness in polynomial time with non negligible probability.

**D as \( \tilde{X} \):** Assume any dishonest party, \( \tilde{X} \), i.e., \( \tilde{X} \) does not know \( s_x \) such that \( (g)^{sx} = \ell \). Let \( Pr^Y_X(\tau(Y, I)) \) denotes the probability that \( \tilde{X} \) outputs 1 indicating that \( Y \) can prove knowledge of \( I \). To establish the second condition for verifier soundness, we prove that: \( |Pr^Y_X(\tau(Y, I)) - Pr^Y_{\tilde{X}}(\tau(\overline{Y}, I))| \leq \epsilon (|I|) \), i.e., \( D \) acting as \( \tilde{X} \) can not distinguish \( Y \) from any other \( Y \).
For CDLL this condition trivially holds, as \( \overline{Y} \) sends only random coins to \( X \), and any \( \overline{Y} \) can send the same random coins to \( X \). Note that the knowledge of a valid \( s_y \), such that \( g^{s_y} = \ell \) is what distinguishes \( \overline{Y} \) from any other \( Y \). Hence there does not exists \( D \) that distinguishes \( \overline{Y} \) from \( Y \).

\[ \blacksquare \]

**Theorem 5.2** CDLL(\( \overline{X}, \overline{Y} \)) is a zero knowledge proof system of membership in a common DL language based on the hardness of DDH.

**Proof** For any \( \hat{C} \), our proof considers two cases for \( \hat{C} \). The first is the case where \( I \) is in the common language of \( \hat{C} \) and \( \overline{X} \) or \( \overline{Y} \), but \( \hat{C} \) may exhibit arbitrary behavior. In that case, we show a simulation of \( \overline{X} \) using the knowledge that \( \hat{C} \) has of \( I \), and a simulation of \( \overline{Y} \) using the knowledge that \( \hat{C} \) has of \( I \). The second case, is when \( I \) is not in the common language of \( \hat{C} \) and \( \overline{X} \) or \( \overline{Y} \).

We start by the first case. For all \( \hat{C} \), and for any accepting interaction \( [(\overline{X}, \hat{C})(I)] \), \( I \) is in the common language of \( \overline{X} \) and \( \hat{C} \), i.e., \( \hat{C} \) has knoweldge of \( I \). We start by showing a simulation for the honest party \( \overline{X} \) done by simulator \( M_{\hat{C}}(I) \), i.e., \( M_{\hat{C}}(I) \) using the knowledge of \( \hat{C} \) as follows:

**Simulation** \( M_{\hat{C}}(I = (\ell, g, q, p)) \)

- Step 1: Let \( \hat{C} \) send the first message: \( B, n \).

- Step 2: Choose \( \hat{r}, R \in [0,q-1] \), compute \( r \) and derive \( M = g^r \), and \( K = B^R \ell^{-rns} \).

- Step 3: Send \( M, R \) and \( K \).

This is a polynomial time simulation on a given \( I \) which will succeed in convincing any \( \hat{C} \) that it is interacting with \( \overline{X} \) on \( I \) in the common language of both. This simulation will
produce a probability ensemble \( \{ M_{\hat{C}}(I) \}_{I \in \hat{L}_T(X, \hat{C})} \) that is computationally indistinguishable from the probability ensemble \( \{(X, \hat{C})(I)\}_{I \in \hat{L}_T(X, \hat{C})} \).

On the other hand for strategy \( \hat{C} \), and for any accepting interaction \([(\hat{C}, \bar{Y})(I)]\), \( \bar{Y} \) uses only its random coins for deriving \( B \) and \( n \). It then follows that nothing can be leaked from \( \bar{Y} \). Regardless of the state of knowledge of \( \hat{C} \).

Next, we consider \( \hat{C} \) that has no knowledge of \( I \), i.e., \( I \) is not in the common language of \( X \), or \( \bar{Y} \). \( \bar{Y} \) uses only its random coins, so no knowledge can be leaked from \( \bar{Y} \). We now consider the interaction of arbitrary \( \hat{C} \) and honest party \( X \) and show a simulation that uses the strategy of \( \hat{C} \) to simulate \( X \) as follows:

**Simulation** \( M_{\hat{C}}(I = \langle \ell, g, q, p \rangle) \)

- **Step 1:** Let \( \hat{C} \) send the first message: \( B \), \( n \).

- **Step 2:** Choose \( \hat{r} \), compute \( r \), and derive \( M = g^r \)

- **Step 3:** If the exponent \( b \) of \( B \) is known, derive \( K = B^{r \ell - rnb} \). Else, skip to the next step.

- **Step 3:** Send \( M \) and randoms for \( R \) and \( K \).

Based on the proven verifier soundness, \( \hat{C} \) can not distinguish this simulation from an honest interaction with \( X \). Hence, no knowledge is leaked from \( X \) to \( \hat{C} \).
5.2 CLHG Protocol for Identification based on Proofs on a Common Graph Hamiltonian Language

The CLHG protocol in Table 5.2 assumes that both $X$ and $Y$ knows the same Hamiltonian cycle of graph $G$. The interaction starts on common input $G$, and the first message is sent from $Y$ and it contains a permuted cycle $B_1$, a permutation $\eta_2$, and graph $G_1 = \eta_1 \eta_2(G)$. $X$ then uses its cycle $C$ to derive $\eta_3$ from $B_1$. Next $X$ derives $G' = \eta_3 \eta_2(G)$ and checks that $\eta_3$ is a valid isomorphism by checking that $G' = G_1$ and that $\eta_3 \eta_2(C)$ is a cycle in $G_1$. If not, then it generates random $R$ in which $B_1$ is a cycle, otherwise it computes $R = \eta_3(G)$. The verification at $Y$ checks that $\eta_1(G) = R$. The protocol is a proof of membership in the common language $HML1-\hat{L}_{T_e}(X,Y)$ specified in Table 4.8. The CLHG protocol and security analysis assumes the hardness of finding Hamiltonian cycles in graphs, as well as the hardness of the decidability of the graphs isomorphism problem. I.e., given any two graphs, there is no efficient algorithm that can decide if these graphs are isomorphic or not.

The protocol involves operations on the incidence matrix $\phi(G)$ of graph $G$. This is simply a matrix that completely specifies $G$ as follows: If $G$ has $n$ vertices, then $\phi(G)$ is an $n \times n$ matrix, such that the $(i,j)$’th entry is 1 if there is an edge in $G$ from vertex $i$ to vertex $j$, and is 0 otherwise. Note that if we have a permutation $\eta$ on $n$ elements and apply it to $G$, we can obtain the incidence matrix of $\eta(G)$ easily from $\phi(G)$ by permuting the rows and columns of $\phi(G)$ according to $\eta$. The resulting matrix is called $\eta(\phi(G))$. Note that a Hamiltonian cycle $C$ in $G$ can be specified by pointing out a set of $n$ entries in $\phi(G)$ such that all the entries are 1, and such that they specify a cycle that visits every node exactly once.
<table>
<thead>
<tr>
<th>Party X (C)</th>
<th>Party Y (C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I = \langle G \rangle$</td>
<td>$I = \langle G \rangle$</td>
</tr>
</tbody>
</table>

Choose $\eta_1, \eta_2$

$B_1 = \eta_1(C)$

$G_1 = \eta_1 \eta_2(G)$

$B_1, \eta_2, G_1$

Derive $\eta_3$ s.t.:

$\eta_3(C) = B_1$ and

Derive $G' = \eta_3 \eta_2(G)$ s.t.:

Check:

1. $G' = G_1$

2. $\eta_3 \eta_2(C)$ cycle in $G_1$

If not, create random $R$ s.t. $B_1$ cycle in $R$

Else

$R = \eta_3(G)$

$\overset{\rightarrow}{R}$

Validate:

$\eta_1(G) = R$

Table 5.2: CLHG Protocol for Identification based on a Common Graph Hamiltonian Language for Identical Prover Strategies
Theorem 5.3  \(\text{CLHG}(\overline{X}, \overline{Y})\) is a proof system of membership in a common Graph Hamiltonian language assuming the intractability of finding Hamiltonian Cycles and the intractability of the graphs isomorphism decidability problem.

Proof  Assume negligible function \(\varepsilon : N \rightarrow [0, 1] : \varepsilon(n) = 2^{-n}\), We show that CLHG satisfies Completeness, Prover and Verifier Soundness in that order.

Completeness  To prove completeness, we show that an honest prover that knows the Hamiltonian Cycle \(C\) in Graph \(G\) and \(\overline{G}\), succeeds in satisfying the verification condition with probability 1.

\[
\eta_1(G) = \eta_3(G) \quad \text{(since the correct } C \text{ is known to } X) = R
\]

Prover Soundness:  To establish prover soundness, assume \(\overline{X}\) does not know the Hamiltonian Cycle \(C\) of Graph \(G\). Assume that \(\overline{X}\) cheats by guessing \(\eta_2\) correctly, then it can successfully formulate \(R\). Nonetheless, if \(\overline{X}\) can guess \(\eta_2\) correctly with non negligible probability, then \(\overline{X}\) can successfully derive \(C\) with non negligible probability, i.e., \(\overline{X}\) knows \(C\), which contradicts the initial assumption that \(\overline{X}\) does not know \(C\).

We show that if there exists a probabilistic polynomial time algorithm \(A\) that breaks prover soundness for \(\text{CLHG}\), i.e., convinces \(\overline{Y}\) that \(I\) is in a common language with non negligible probability, then we construct a probabilistic polynomial time algorithm \(\hat{A}\) that breaks the Graph Isomorphism decidability problem. \(\hat{A}\) takes in two graphs \(G_i\) and \(G_j\) as
well as a cycle $c_j$ in $G_j$. $\hat{A}$ outputs 1 if $G_i$ is isomorphic to $G_j$ and outputs 0 otherwise.

Under the intractability assumptions for the Graph isomorphism decidability, an efficient solver exists only if the isomorphism between the two given graphs is known. Let $G$ be $G_i$, then the construction for the solver will then be as follows:

$\hat{A}(c_j, G_i, G_j)$

- Let $G = G_i$.
- Choose random $\eta_2$, send $B_1 = c_j$, and $G_1 = \eta_2 G_j$
- Receive $R$.
- Return 1, i.e., $G_i$ isomorphic to $G_j$ if $R = G_j$, else, return 0.

If $A$ succeeds with non negligible probability in breaking CLHG prover soundness, then $\hat{A}$ will be able to successfully determine if $G_i$ and $G_j$ are isomorphic. Note that honest strategy $X$ can not be fooled by this strategy because of the check it performs, which $\tilde{X}$ can not do as it does not know a cycle of $G$. The construction $\hat{A}$ is an efficient one, and is guaranteed to succeed with non negligible probability if adversary $A$ succeeds in impersonating $X$ at $Y$ with non negligible probability. If $G_i$ and $G_j$ are isomorphic, then there exists an isomorphism $\pi$ such that $G_j = \pi G_i$, and if a cycle $c_j$ is known in $G_j$, then it is not possible to get the cycle of $G_i$ without knowing $\pi$. $\hat{A}$ sets $G$ to $G_i$, the graph for which it does not know a Hamiltonian cycle. Then $\hat{A}$ sends the cycle $c_j$ of $G_j$ and uses a permutation $\eta_2$ to derive $G_1$. Upon receiving these values, $A$ derives $\eta_3 = \pi$ if $G_i$ and $G_j$ are isomorphic, and if not, $\eta_3$ should be an invalid permutation. $A$ derives $R = \eta_3(G) = \pi(G) = G_j$.

Hence, we conclude that since there does not exists $\hat{A}$ that breaks the graph isomorphism decidability in polynomial time with non negligible probability, then there does not exist $A$ that breaks CLHG soundness in polynomial time with non negligible probability.
Verifier Soundness:  To prove verifier soundness, we shall consider all polynomial time strategies $D$ and black box interactions with $X$ and $\tilde{X}$ or $Y$ and $\tilde{Y}$. In the distinguishing experiment of $X$, $D$ impersonates $Y$ to $X$, i.e., it can be considered as $\tilde{Y}$, since it does not have knowledge of a Hamiltonian cycle $C$ of $G$. On the other hand, in the distinguishing experiment involving $Y$, $D$ impersonates $X$ to $Y$, i.e., it can be considered as $\tilde{X}$, since it does not have knowledge of a Hamiltonian cycle $C$ of $G$. (A valid $C$ is one that is known by both honest parties in an interaction.)

$D$ as $\tilde{Y}$:  We show that if there exists a probabilistic polynomial time algorithm $D$ that breaks the first condition of verifier soundness for CLHG, i.e., $D$ can distinguish $X$ from any other $X$ on a given $I$ ($|Pr_D^X(\tau(X, I)) - Pr_D^{\tilde{X}}(\tau(\tilde{X}, I))| > \varepsilon(|I|)$), then we construct a probabilistic polynomial time algorithm $\hat{A}$ that breaks the graph Isomorphism decidability assumption as follows:

$\hat{A}(c_j, G_i, G_j)$

- Let $G = G_i$.
- $D$ sends $\eta_2$, send $B_1$, and $G_1$.
- Use $c_j$ to derive $\eta_3$ and then derive $R$ and send it to $D$.
- Based on the output of $D$, return 1, i.e., $G_i$ isomorphic to $G_j$ if $D$ outputs 1, and return 0 otherwise.

If $D$ succeeds with non negligible probability in breaking the first condition of verifier soundness for CLHG, then $\hat{A}$ will be able to successfully determine if $G_i$ and $G_j$ are isomorphic. The construction $\hat{A}$ is an efficient one, and is guaranteed to succeed with
non negligible probability if adversary $D$ succeeds in impersonating $Y$ with non negligible probability, i.e., if it can distinguish $X$ from $X$ with non negligible probability. If $G_i$ and $G_j$ are isomorphic, then there exists an isomorphism $\pi$ such that $G_j=\pi G_i$, and if a cycle $c_j$ is known in $G_j$, then if $B_1 = \pi_1 C$, then $\hat{A}$ derives $\pi_1 \pi$ as the permutation, and uses it to derive $R = \pi_1 \pi G_j = \pi_1 G_i$. Then if $\pi$ is a valid isomorphism, and $D$ can successfully distinguish $\overline{X}$ from $X$ with non negligible probability, then its output can be used to decide the isomorphism of $G_j$ and $G_j$.

Note that an honest prover will always send a random graph for which $B_1$ is a cycle if its test on the values $B_1$ and $G_1$ fails. I.e., for the case where $D$ fails to send correct values for these, $\overline{X}$ does not reply with correctly computed $R$, i.e., behaves exactly as some dishonest strategy $\tilde{X}$. It then follows that $D$ can not distinguish between $X$ and $\tilde{X}$ when it sends inconsistent values for $B_1$, $\eta_2$ and $G_1$, because they are simply identical. That is why we considered in the above reduction the case where $X$’s behavior is different than $\tilde{X}$, i.e., the case where the values for $B_1$, $\eta_2$ and $G_1$ pass the check at $X$.

Hence, we conclude that since there does not exists $\hat{A}$ that breaks the graph isomorphism decidability in polynomial time with non negligible probability, then there does not exist $D$ that breaks the first condition of CLHG verifier soundness in polynomial time with non negligible probability in its experiment with $X$.

**$D$ as $\tilde{X}$:** For the second condition of verifier soundness, it suffices to note that any $Y$ can send random values for $B_1$, $\eta_2$ and $G_1$ and be indistinguishable from $\overline{Y}$, especially that if $D$ uses $\overline{X}$ to detect a difference in the reply, it will still not be able to distinguish, as $\overline{X}$ always replies with a graph for which $B$ is a cycle, regardless, or not if its the correct graph. Based on the proved first condition of verifier soundness, no efficient strategy can
distinguish the reply of $\overline{X}$ from $\overline{\tilde{X}}$. Also the values of the message sent by $\overline{Y}$ and $Y$ are indistinguishable, based on the hardness of finding the isomorphism, and also the hardness of finding a Hamiltonian cycle. Hence, $D$ can not distinguish $\overline{Y}$ from any other $Y$ on a given $I$ except with negligible probability. I.e., $|\left(Pr^Y_D(\tau(Y,I)) - Pr^\overline{Y}_D(\tau(\overline{Y},I))\right)| \leq \varepsilon(|I|)$.

Theorem 5.4 \( CLHG(\overline{X},Y) \) is a zero knowledge proof system of membership in a common Graph Hamiltonian language assuming the intractability of finding Hamiltonian Cycles and the intractability of the graphs isomorphism decidability problem.

Proof For any $\hat{C}$, our proof considers two cases for $\hat{C}$. The first is the case where $I$ is in the common language of $\hat{C}$ and $X$ or $Y$, but $\hat{C}$ may exhibit arbitrary behavior. In that case, we show a simulation of $\overline{X}$ using the knowledge that $\hat{C}$ has of $I$, and a simulation of $\overline{Y}$ using the knowledge that $\hat{C}$ has of $I$. The second case, is when $I$ is not in the common language of $\hat{C}$ and $\overline{X}$ or $\overline{Y}$.

We start by the first case, i.e., when $I$ is in the common language of $\hat{C}$ and $\overline{X}$ or $\overline{Y}$. The proof then follows from the fact that both knows the same cycle $C$ of $G$. It then follows that for all $Y$, and for any accepting interaction $[(\overline{X}, Y)(I)]$, an efficient simulation can be constructed using the knowledge of $Y$ of the cycle $C$. The simulation follows the protocol of $\overline{X}$. Similarly, for all $X$, and for any accepting interaction $[(X, \overline{Y})(I)]$, the simulation uses the cycle known to $X$ and follows the protocol of $\overline{Y}$.

Next, we consider $\hat{C}$ that has no knowledge of $I$, i.e., $I$ is not in the common language of $\overline{X}$, or $\overline{Y}$. We first consider the interaction of arbitrary $\hat{C}$ and honest party $\overline{X}$ and show a simulation that uses the strategy of $\hat{C}$ to simulate $\overline{X}$ as follows:

Simulation $M_{\hat{C}}(I = G)$

- Step 1: Let $\hat{C}$ send the first message.
• Step 2: Send random $R$ such that $B_1$ is a cycle of $R$.

Based on the proven verifier soundness, $\hat{C}$ can not distinguish this simulation from an honest interaction with $X$. Hence, no knowledge is leaked from $X$ to $\hat{C}$.

Similarly, the message of $Y$ can be easily simulated by $\hat{C}$ by sending randoms. and based on verifier soundness, the simulation is computationally indistinguishable from the interaction with $Y$.

5.3 Applications

In this section we propose a number of applications for proofs on common languages that use identical prover strategies, particularly those that wish to eschew the use of identities for privacy preservation. We propose protocols based on the CDLL protocol to demonstrate the power of this simple form of proofs of membership on common languages. The CDLL protocol designates an efficient zero knowledge solutions for several applications and it compares favorably to the traditional solutions deployed for these applications.

5.3.1 An Efficient Signature Scheme based on CDLL

The Fiat-Shamir transform has been used for deriving efficient and secure signature schemes from existing identification schemes based on interactive proofs. The signer is mapped to the prover of the ID scheme, hence, it is given public and secret keys ($pk, sk$) of the prover. To sign a message $M$ it computes the commitment $C = f(c)$, for some one way function on some random $c$ just as the prover would, hashes $C$ and $M$ using a public hash function $H: \{0,1\} \to \{0,1\}$, to obtain a “challenge” $n = H(C||M)$, computes a response $R$ just as the prover would, and sets the signature of $M$ to $C \parallel R$. To verify that $C \parallel R$
is a signature of \( M \), we compute \( n = H(C \parallel M) \) and then checks that the verifier of the identification scheme would accept, namely \( V(pk, C \parallel R) = 1 \). The ZK property of the resulting protocols is modeled by a simulator that is given full control over \( H \). In the real world, the prover and verifier evaluate the hash \( H \) by submitting an input \( x \) to an “oracle" and obtaining the response \( H(x) \). In the “ideal world", we let the simulator take control of this oracle, so that when the verifier inputs \( x \) to the oracle, the simulator decides what value to return as long as the same input results in the same output.

From the above description, it is clear that the transform is based mainly on finding an alternative to the verifier’s \( V \) interaction by substituting \( V \)’s challenge \( n \) by a one way function. Assuming the existence of strong collision resistant hash functions, given a certain \( n \), it is hard to determine the input to the hash function that would give this \( n \). Recall the standard ZK attack in which a correct guess for \( V \)’s challenge \( n \) can enable a dishonest prover to impersonate an honest prover without knowing the secret \( s \). Hence the importance of linking the commitment to the challenge through the hash function, which guarantees that the choice of the commitment always precedes the choice of the challenge since the hash functions disallows the opposite.

In Table 5.3 we propose a non-interactive ZK protocol based on proofs on Common Languages (CLSDL). The protocol does not directly use the FS transform:

- The FS transform guarantees that any verifier can verify the signature, which justifies the use of \( g^a \) as a parameter to the hash function. On the other hand, based on verifier soundness, not all verifiers can verify the signature.

In our Common Language Signature protocol (CLSDL), \( g^a \) has to be replaced by a value that can only be derived by honest verifiers and honest provers. Accordingly, we replaced “\( g^a \)” by “\( a \)”.

108
• The hash function in the FS transform is used to ensure that the prover commits to a value before knowing the challenge which imitates the order implied by the three message ZK interaction. In the CLSDL protocol, the verifier starts by sending $g^b$ and $n$ and the prover replies, i.e., this is not a concern for CLSDL.

Nonetheless the use of $g^b$ sent by $V$ and $g^r$ sent by $P$ is to apply a one way function on the value of “$a$” extracted from $R = ns + a$, this can be replaced by the use of a hash function on $a$. Moreover, the CBSD protocol chooses the value of $a$ using its random tape along with the value of $n$ to avoid reusing randoms for different challenges. This can also be guaranteed by setting $n = H(a)$, which gives the same $n$ when the same $a$ is used.

To sign a message, $m$, it would be included in the hash along with “$a$” as shown in the protocol. However, in this case, the value of $a$ should be derived using the random tape along with $H(m)$, to guarantee that no two different signatures are produced using the same $a$.

To proof the ZK property for this protocol, the simulator needs only to send random values for the signature. If the verifier submits a query on “$a$”, then the simulator checks if $a$ is a valid solution to $R - ns$, where $n = h(a || m)$, if it was, then the simulator does not return a value to the verifier, and it uses $a$ to derive $s$ and rewind the verifier to simulate using $s$. Otherwise, the simulator returns a value that is not equal to the random hashed value “$H$” sent in the message. In this case, the simulator knows that the verifier has no knowledge of $s$, and would fail to verify even with an honest prover.

The proposed protocol uses simpler and more efficient constructs than protocols using the Fiat Shamir transform.
<table>
<thead>
<tr>
<th>Signer ( (X) )</th>
<th>Verifier ( (Y) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_X = s )</td>
<td>( (s_X = s) )</td>
</tr>
</tbody>
</table>

\[
I = \langle \ell, g, q, p \rangle, \\
\text{where } \ell = g^s
\]

\[
n \in_u [0, q - 1] \\
a = \text{rand}([0, q - 1], h(m)) \\
R = h(a|m)s + a \\
H = h(a|m)
\]

\[
m \cdot H||R \\
\text{If } s \text{ is known} \\
a = R - Hs \\
\text{validate:} \\
H = h(a|m)
\]

Table 5.3: Signature Scheme based on a Common DL Language (CLSDL)

### 5.3.2 CDLL for Symmetric Key Protocols Simplify Management

Most existing symmetric key protocols are not ZK, their explicit use of keys in communications yields a number of issues associated with key management. For example, in an authentication system such as Kerberos or Shibboleth, correspondences with an Authentication Server use a session key derived from the user’s password, which makes it essential to continuously update the password to avoid related key attacks.

Symmetric keys trivially yield a common language between two parties. The use of CDLL for these interactions protects the passwords since the protocol is zero knowledge. This is efficiently achieved by the lightweight discrete log computations and the use of two messages (note that most symmetric key based interactions take the form of challenge-response messages).
5.3.3 CDLL for Deniable Authenticated Key Exchange

We demonstrate an important application of CDLL for authenticated key agreement (AKE) with little or nearly no overhead. Our ZK protocol allow both the prover and the verifier to contribute information from which a joint key is established. This property comes as part of the protocol without any further additions. Alternatively, existing ZK schemes does not naturally support this feature. This implies that AKE comes with no overheads through CDLL but rather as an additional potential application.

We study the use of CDLL ZK protocol for deniable AKE based on a symmetric secret, $s$. We show that the CDLL protocol allows $X$ and $Y$ to agree on a new key that satisfies important AKE properties. The use of CDLL for AKE adds no overhead and does not involve extra exchanges or rounds, but rather preserves the secrets for lifetime backed by the zero knowledge property while ensuring strong decoupling of the key and the secret. Currently we established that our AKE protocol is secure in the CK model [25] under the DDH assumption, and we are working on proving the deniability property. Our future work focuses on studying the protocol in the extended CK model, which implies strengthening the protocol against a new set of attacks that are inherent in the use of symmetric secrets but can be circumvented by using the established Common language. Our current AKE protocol is secure assuming strong collision resistant hash function (i.e., in the Random Oracle model), hence we study the possibility of extending the protocol to the standard model.

**Background:** AKE protocols are expected to satisfy certain security requirements that characterize the class of applications where they are employed. These have led to the development of a number of formal security models of AKE, including the Canetti-Krawczyk
model (CK) [25]. The CK model combines the proofs of confidentiality as well as authentication in a modular fashion through two main adversarial models: the idealized authenticated links model (AM) and the unauthenticated links adversarial model (UM). In the AM model an attacker is restricted to only deliver messages truly generated by the parties without any change or addition. Whereas in the UM model the adversary has full control of the communication links. The basic protocol underlying our AKE variants is the simple 2-step DH protocol. This has been proven secure in the AM model in [25]. Our ZK protocols uses the basic DH structure and assumes the CK adversary capabilities in both the AM and UM models.

The CK model assumes that there are \( \tilde{n} \) parties, \( P_1 : P_{\tilde{n}} \), each running a copy of a message-driven protocol. The computation consists of a sequence of activations of this protocol within different parties, each of which is called a session and is identified by a unique session id (\( sid \)). The activations are controlled and scheduled by the adversary which also decides which incoming message or external requests the activated party is to receive. Let \( \hat{P} \) be the set of active parties and let \( \hat{M} \) be the set of undelivered messages that are sent during the CK experiment. A newly activated party is added to \( \hat{P} \). The AM model adversary \( U \) can only use a message from \( \hat{M} \) to activate or communicate with any party \( P_i \). This message must have \( P_i \) specified as a recipient or else \( U \) cannot send it to \( P_i \). After a message is sent it is deleted from \( \hat{M} \). On the other hand, the UM adversary \( A \) can communicate with any party \( P_i \) using messages that are not in \( \hat{M} \). This is the main difference between the two adversaries, otherwise they are identical and both can issue the following queries:
• corrupt($P_i$): $A$ may corrupt $P_i$ and learn the entire current state of $P_i$ including any long term secrets, session internal states and session keys. $A$ can then impersonate $P_i$ to any other entity.

• session-key($P_i$, $sid$): This returns the session key accepted by $P_i$ if any for a given session $sid$.

• session-state($P_i$, $sid$): This returns all the internal state of party $P_i$ associated with a particular incomplete session $sid$. This does not include the local state of the sub-routines that directly access the long-term secret information, e.g. the local signature/decryption key of a public-key crypto system, or the long-term shared key.

• test-session($P_i$, $sid$): For this query, a random $m \in \{0, 1\}$ is chosen. If $m=1$ then the session key is returned, otherwise a random key is returned. This query can only be issued to a session that has not been subject to any of the previous three queries.

A game is setup, where an adversary can choose where to submit the test-session query, i.e., to which party and for what session. Then no further queries can be submitted to this session and no further test-session can be made to any other entity. Then eventually the adversary outputs its guess for $m$. The notion of Session-Key (SK) security defines security in the CK model as follows:

**Definition 5.1** A key-agreement protocol is Session-Key secure (or SK-secure) if the following properties hold for any adversary $A$ in the UM model:

• If two uncorrupted parties complete matching sessions then they both output the same key, and
• The probability that $A$ can distinguish the session key from a random value is no more than $1/2$ plus a negligible fraction in the security parameter.

**CDLL for AKE (ZK-CAKE)** Let $X$ and $Y$ be two parties sharing a secret $s$. It is required that $s$’s secrecy is preserved on a lifelong basis, and that $X$ and $Y$ derive new shared secrets/keys based on $s$. The CDLL protocol uses a value, $K$, derived by $X$ and verified by $Y$ using the shared knowledge of $s$. $\hat{K}$ is computed using $K$ which is formed of three key parts, $g^r$, $g^a$ and $g^b$, one of which is not sent; $g^a$. The ability to derive $K$ along with the ZK response authenticates $X$ to $Y$ and the ability to validate $K$ implicitly authenticates $Y$ to $P$, since we have shown that $Y$ that does not know $s$ it can not derive $K$ and hence can not validate $\hat{K}$. We assume that $s$ is unique for any $X$ and $Y$. When the CDLL protocol is used for AKE, both $X$ and $Y$ derive the session key $\hat{K}$, but $\hat{K}$ should not be sent in the clear. $g^r$ serves also to support forward secrecy of the session keys, since it guarantees that if $s$ gets leaked, past session keys will not be compromised. The ZK-CAKE protocol is shown in Table 5.3.3 and proof of security in the Canetti-Krawczyk (CK) model [25] with perfect forward secrecy can be found in the appendix. The protocol uses the CK model [25] notations detailing a key exchange session $sid$ between $P_j$, the session initiator, and $P_i$ the session responder. A message is denoted by a tuple $(sender, sid, msg)$.

**Theorem 5.5** ZK-CAKE is SK-secure with perfect forward secrecy assuming the intractability of DDH and assuming the existence of strong collision resistant hash functions.

**Proof** Let $A$ be a polynomially bounded UM adversary against ZK-BAKE, that runs in time $t(\beta)$, activates at most $m(\beta)$ honest parties, at most $i(\beta)$ sessions and makes at most $\tilde{h}(\beta)$ queries to the oracle $h$, where $t(\beta), m(\beta), i(\beta)$ and $\tilde{h}(\beta)$ are polynomially bounded in $\beta$. Let $A$ be an adversary that has a non negligible success probability $v(\beta)$ in UM model against ZK-BAKE. We show that if such $A$ exists then we can construct a DDH solver.
Protocol ZK-CAKE

- The initiator $P_j$ on input $(P_j, P_i, sid)$, chooses $b \in [0, q - 1]$ and $n \in [0, q - 1]$ then sends $(P_j, sid, n, g^b)$ to $P_i$.
- Upon receipt of $(P_j, sid, n, g^b)$, the responder chooses $a \in [0, q - 1]$ and $r \in [0, q - 1]$, computes $g^a$ and $k = g^{arb}$ and the ZK response $\theta = ns + a$, erases $a$ and $r$, and then computes $K = h(k, g^a, g^b, g^r, n)$, erases $k$, outputs the session key under session-id $sid$ and sends the ZK response $\theta$ along with $g^r$ and $g^b$ to $P_j$ and then sends $(P_i, sid, g^r, \theta)$ to $P_j$.
- Upon receipt of $\theta$, $P_j$ verifies the ZK response. If the verification succeeds it derives $a$ from $\theta$ using $s$, computes $k = g^{arb}$, erases $a$, $b$ and computes $K = h(k, g^a, g^b, g^r, n)$, erases $k$ and outputs the session key $K$ under session-id $sid$.

Table 5.4: ZK-CAKE

$DD$ to output true for any input $(X = g^x, Y = g^y, Z)$ if and only if $Z = g^{xy}$, otherwise it outputs false. We shall model the hash function $h(.)$ as a public random oracle.

$DD$ should simulate the CK experiment such that $A$ can not distinguish the simulation from the real CK experiment. $DD$ simulates the $m(\beta)$ parties and answers all queries made by $A$. $DD$ answers any hash queries by returning a random value, which is then stored in a table associated with the used hash function and maintained by $DD$.

Since the protocol assumes a shared $s$ between any two entities interested in establishing a common secret, $DD$ assigns a unique $(s, g^x)$ for every distinct pair $(P_i, P_j)$ such that every entity can establish a key with every other entity. $DD$ starts the simulation and randomly picks a session $sid*$ to be the test session, we let $P_i*$ and $P_j*$ be the entities in the chosen session. $DD$ then starts $A$.

For $sid*$, $DD$ does not follow the protocol fully, instead, for $P_i*$ it sets $g^a$ to $X$, and for $P_j*$ it sets $g^b$ to $Y$. During the simulation, any messages generated by $DD$ is placed in $\hat{M}$ for $A$ to manage their delivery. $DD$ terminates $A$ and aborts the simulation if $A$ tries to expose
by issuing any of the queries: session-key, session-state or corrupt on any of the parties of sid* or if A issued a test-session query on sid ≠ sid* or if session sid* is invoked within a party Pi ≠ Pi*. The test-session query on sid* will be replied by a random key. Otherwise queries made by A during the simulation can be answered by DD as follows:

- When A activates a party to send or receive a message in some sid ≠ sid*, DD follows the ZK-CAKE for generating the needed randoms of Pi and simulates the response of Pj as per the protocol as well. All undelivered messages are placed in ˆM. This simulation is easily done since DD knows all the long term secrets and the randoms used in sid by any of these parties.

Remark: For any session, including sid* the UM adversary A is allowed to impersonate Pi to Pj or vice versa during the simulation. Nonetheless, the proofs of soundness of CDLL have shown that a prover or a verifier can only succeed with negligible probability in deriving a valid K without knowing s. Hence mutual authentication is implicitly achieved and impersonation can only succeed with negligible probability in polynomial time.

- Exposures to any sid ≠ sid* can be easily handled even when any of Pi* and Pj* are involved as one of the responders. This is because the private keys are known to DD and a shared s is unique for every (Pi, Pj) pair. Hence DD can answer all the queries as per the protocol using the randoms and the private secrets of the parties of sid.

- When sid* expires, A is allowed to corrupt Pi* and/or Pj*. ZK-CAKE ensures that any ephemeral secrets are deleted once they have been used such that when the session expires an entity’s internal state contains only its long term secrets. Hence DD
can easily reply the corrupt query on any of the test session entities by returning the private secrets that is well known to it.

A cannot issue any of the queries on any session that matches sid*. It can not also perform key replication attacks using a non-matching session that produces the same session key as the test session, since non-matching sessions either have different entities, and hence a different value for $g^i$, or different session values for any of $n$, $g^a$, $g^b$ or $g^r$, all of which produce a different session key. And since hash functions are modeled as a random oracle, the probability of guessing the output of $h()$ for $K$ is negligible in $t(\beta)$.

A could also attempt a forging attack by providing a correct guess for $g^{arb}$ of the value of $k$, but then this would immediately allow DD to solve for $Z$, since A would need to query $h(.)$ on the key tuple that contains $k$. DD can then use $k$, check if $k=Z$ and return true or false accordingly. Finally when sid* expires, A can corrupt $P_i*$ and $P_j*$, learn $s$ and derive $a$ from the ZK response of sid*. But then in order to correctly derive $K$, A would still need to successfully compute $k$ and query the key tuple containing $k$. But this would again allow DD to check for $Z$ using $k$ as shown before.

The above simulation is indistinguishable from the real CK experiment based on the indistinguishability of the ZK instances as well as the DDH assumption. The simulation demonstrates that if A succeeds with non negligible probability in this simulated CK experiment, then it is also possible to solve the DDH problem with non negligible probability, which contradicts the hardness assumptions of the protocol.

5.3.4 CDLL for Zero Knowledge Secret Handshakes

A secret handshake is a privacy-preserving mutual authentication wherein two parties establish a secure, anonymous and unobservable communication if and only if they are
members of the same group [6]. In contrast to existing secret handshake protocols that base their authentication on the identities/credentials of the parties involved, secret handshake protocols based on CDLL would have entities prove that some value \( I \) is in a common language. To achieve strong privacy, entities would keep \( \ell \) private and just publish \( p \) and \( q \).

The protocol would closely follow the one discussed for AKE except that a third message would be required from \( Y \) to \( X \) to confirm the handshake to \( X \); we note that a third message is also used for traditional handshake protocols to confirm successful completion of the handshake. Secret handshakes are concerned with not leaking the affiliation of the entities, but they do not constrain what each of the entities can learn from each other. CDLL refinement would add zero knowledge to secret handshakes, as well as satisfy other desirable properties in terms of anonymity and unlinkability.
Chapter 6: Non-Identical Prover Strategies for Common Languages

In this chapter we study protocols for proofs on the common languages defined for non-identical prover strategies in Chapter 4. We consider proofs on the common discrete log language $DL_2\hat{L}_{T^r}(X,Y)$, the common RSA language $RSA_1\hat{L}_{T^r}(X,Y)$ and the common Graph Hamiltonian language $HML_2\hat{L}_{T^r}(X,Y)$. All the protocols constitute an interaction on a public common input $I$ that is not certified and can be proposed by any of the two parties $X$ or $Y$. The interaction on the common input constitutes a proof of membership of $I$ in the corresponding common language.

6.1 A-CDLL Protocol for Identification based on a Common Discrete Log Language

In this section we present A-CDLL, a two party interactive protocol for identification based on membership in a Common Discrete Log Language (CDLL) that admits non identical prover strategies at the two parties involved. The protocol is based on the common language specification listed in Table 6.1, and assumes the intractability of Decision Diffie-Hellman for the defined group $G$.

Similar to CDLL, the key generation center (KAC) generates two primes $p$ and $q$, where $q$ is a prime factor of $p - 1$, i.e., $q|\hat{p}(p - 1)$, $q \geq 2^{140}$, $p \geq 2^{512}$. Parties do not need a secure
connection with KAC, instead, they just need to publicly contact KAC to obtain \( p, q \), all of which can be sent in public.

We assume the interaction is for proving membership of a given \( I = \langle \ell, W, g_y, g_x, q, p \rangle \) in the common language \( DL2-\hat{L}_{T_e}(\overline{X}, \overline{Y}) \) of \( X \) and \( Y \). Note that the KAC does not need to certify any of the values that are publicly sent. The A-CDLL protocol uses two messages. The first is sent from \( Y \) to \( X \) and contains a DH key part \( B = (g_y)^b \) as well as a random \( n \). This message also contains a generator \( g \) and value \( \ell \). The second is sent from \( X \) to \( Y \), and contains a DH key part \( M = (g_x)^r \), and a DH key \( K \) derived using \( n, r, s_x \) and \( B \). If \( X \) detected that \( (\hat{g})^{s_x} \) is not equal to \( \ell \), it sends random values for \( M \) and \( K \). \( Y \) uses the verification expression \( K = (\hat{\ell})^{(n_{1b})(M)^{(s_yf)}} (mod \ p) \) to check if \( I \) is in \( DL2-\hat{L}_{T_e}(\overline{X}, \overline{Y}) \). This verification can only succeed if and only if for both \( X \) and \( Y \), \((g_y)^{s_x} = (g_x)^{s_y} \). The protocol is described formally in Table 6.1. Note that all equalities or assignments are \( mod \ p \).

**Theorem 6.1** A-CDLL\((\overline{X}, \overline{Y})\) is a proof system of membership in the common language \( DL2-\hat{L}_{T_e}(\overline{X}, \overline{Y}) \).

**Proof** Assume negligible function \( \varepsilon : N \rightarrow [0, 1] : \varepsilon(n) = 2^{-n} \). We prove Completeness, Prover and Verifier Soundness in that order.

**Completeness:** \( X \) successfully responds to \( Y \)’s challenge using its \( s_x \) and private DH exponent \( r \) to formulate \( K \).

If \( Y \) knows \( s_y \), such that \( (g_x)^{s_y} = \hat{\ell} \), it successfully verifies:

\[
((M)^{(s_yf)}(\hat{\ell})^{(n_{1b})}(W)^{(n_{2b}))}) = ((g_x)^{rs_yf}(g_x)^{n_{1bs_y}}(g_y)^{n_{2b}/s_y})(mod \ p) = (g_x)^{rs_yr}(g_x)^{s_xn_{1b}(g_x)^{n_{2b}/s_x})(mod \ p) = (\hat{\ell})^{(r)(g_y)^{s_xn_{1b}(B_x)^{n_{2}/s_x})(mod \ p)} = (\hat{\ell})^{(r)(g_y)^{bn_{1s_x}(B_x)^{n_{2}/s_x})(mod \ p)}
\]
\( I = \langle \hat{\ell}, W, g_y, g_x, q, p \rangle \)
where \( \hat{\ell} = (g_y)^{s_x} \)
and \( W = (g_x)^{(1/s_x)} \)

\begin{align*}
\ell, \hat{\ell}, \hat{\ell} &\in \{0, q - 1\} \\
\hat{\ell} &=(g_x)^{s_y} \\
\hat{\ell} &=(g_y)^{s_y} \\
B_y &=(g_y)^b \\
B_x &=(g_x)^b \\
\text{Check:} \\
1. & \hat{\ell} = \hat{\ell} \text{ (set } F = 1) \\
& \text{ or } \hat{\ell} = \hat{\ell} \text{ (set } F = 2) \\
2. & W = (g_y)^{(1/s_y)} \text{ or } (g_x)^{(1/s_y)} \\
\text{If 1 or 2 fails, send random } \hat{\ell}, \hat{g}. \\
\text{Else} \\
\ell &=(\hat{\ell})^f \\
\hat{g} &=(g_y)^f \text{ if } F = 1 \\
\text{Else } \hat{g} &=(g_x)^f \\
\ell, \hat{g}, B_x, B_y, n_1 &\rightarrow \text{ Validate} \\
\end{align*}

Check:
\( \ell = (\hat{g})^{s_x} \)
If not, send random \( M, K \)
Else:
\( \hat{r}, n_2 \in \{0, q - 1\} \)
\( r = \text{rand}(\hat{r}, \ell, \hat{g}, B, n) \)
\( M = (g_x)^r \)
\( K = (\ell)^r(B_y)^{(n_1s_x)}(B_x)^{(n_2/s_x)} \)
\( M, n_2, K \rightarrow \text{ Validate} \)
\( K = \hat{\ell}^{(n_1b)}W^{(n_2b)}(M)^{(s,f)} \)

Table 6.1: A-CDLL Protocol for Identification based on Proofs on Common DL Language for non Identical Prover Strategies
\[ \ell (r) B^{ns_x} (B_x)^{n_2/s_x} \pmod{p} = K \pmod{p} \]

Alternatively, if \( Y \) knows \( s_x \), such that \((g_y)^{s_y} = \ell \), then \( s_y = s_x \). \( Y \) then successfully verifies:

\[
\left( (M)^{(s_y f)} \right) (\ell) (n_1 b) (W)^{(n_2 b)} \\
= (g_y)^{rs} (g_y)^{n_1 b s_y} (g_x)^{n_2 b / s_x} \pmod{p} \\
= (\ell)^{r} (g_y)^{s_x n_1 b} (B_x)^{n_2 / s_x} \pmod{p} \\
= (\ell)^{r} (g_y)^{b n_1 s_x} (B_x)^{n_2 / s_x} \pmod{p} \\
= (\ell)^{r} B^{ns_x} (B_x)^{n_2 / s_x} \pmod{p} = K \pmod{p}
\]

From the above, we conclude that \( K = \hat{\ell}^{(n_1 b)} W^{(n_2 b)} (M)^{(s_y f)} \). Hence, \( \hat{X} \) succeeds with probability 1 in convincing \( Y \) that \( I \) is in the common language \( DL2 \cdot L_{T \tau} (X, Y) \). This proves completeness with probability 1.

**Prover Soundness:** To establish prover soundness, assume a dishonest party, \( \tilde{X} \), i.e., \( \tilde{X} \) does not know \( s_x \) such that \((g_y)^{s_x} = \ell \) and \((g_x)^{1/s_x} = W \). Based on the assumed hardness of DDH, we prove that \( Y \) accepts the proof of \( \tilde{X} \) as valid with probability bounded by \( \epsilon(|I|) \).

We show that if there exists a probabilistic polynomial time algorithm \( A \) that breaks prover soundness for A-CDLL, i.e., convinces \( Y \) that \( I \) is in a common language with non negligible probability, then we construct a probabilistic polynomial time algorithm \( \hat{A} \) that breaks the CDH intractability assumption.

\( \hat{A}(g, R_1, R_2) \)
• Choose random $u$. Assume $\ell = R_2$, $g_s = (R_2)^u$, $W = g^u$, $g_y = g$

• Randomly choose $f, v$ in $[0, q - 1]$, and random generator $\hat{g}$.

• Send $\ell = (g_s)^f, \hat{g}, B_y = R_1, B_x = (g_x)^v$ and $n_1$ to $A$.

• Receive $M, n_2$ and $K$.

• Return Diffie Hellman key $(KM^{-f}g^{-n_2uv})^{1/n_1}$

If $A$ succeeds with non negligible probability in breaking A-CDLL, then $\hat{A}$ will be able to successfully find the DH key for the Diffie Hellman key parts $R_1$ and $R_2$ and generator $g$. Clearly the construction $\hat{A}$ is an efficient one, and is guaranteed to succeed with non negligible probability if adversary $A$ succeeds in impersonating $\bar{X}$ at $\bar{Y}$ with non negligible probability. $\hat{A}$ sends its message to $A$, and then $A$ replies back by $M = (g_s)^v$, and $K = (g_s)^f R_1^{n_1 s_x} (R_2)^{n_2uv/s_x} = (g_s)^f R_1^{n_1 s_x} (g)^{n_2 uv}$. The value “$R_1^{s_x}$”, constitutes the DH key for $R_1$ and $R_2$. $\hat{A}$ is able to derive this value from $K$.

Based on the assumption that $A$ does not know $s_x$, then $A$ can not perform the check that is normally done by $\bar{X}$, i.e., checking that the discrete log of $\ell$ to base $\hat{g}$ is $s_x$. Hence, to $A$, based on the hardness of DDH, the values of $\ell$ and $\hat{g}$ sent by $\hat{A}$ are indistinguishable from those that are sent by $\bar{X}$.

Hence, we conclude that since there does not exists $\hat{A}$ that breaks the CDH assumption in polynomial time with non negligible probability, then there does not exist $A$ that breaks A-CDLL soundness in polynomial time with non negligible probability.

**Verifier Soundness:** To prove verifier soundness, we shall consider all polynomial time strategies $D$ and black box interactions with $\bar{X}$ and $\bar{X}$ or $\bar{Y}$ and $\bar{Y}$. In the distinguishing experiment of $\bar{X}$, $D$ impersonates $\bar{Y}$ to $\bar{X}$, i.e., it can be considered as $\bar{Y}$, since it does not
have knowledge of the discrete log of \( \ell \) to base \( g_x \) or base \( g_y \). On the other hand, in the distinguishing experiment involving \( Y, D \) impersonates \( X \) to \( Y \), i.e., it can be considered as \( \tilde{X} \), since it does not have knowledge of the discrete log of \( \ell \).

**D as \( \tilde{Y} \):** Assume dishonest party, \( \tilde{Y} \), i.e., \( \tilde{Y} \) does not know \( s_y \) such that \((g_x)^{s_y} = \ell \) and \((g_y)^{(1/s_y)} = W \). Let \( Pr_Y^X(\tau(X,I)) \) denotes the probability that \( \tilde{Y} \) outputs 1 indicating that \( X \) can prove knowledge of \( I \) after interacting with \( X \). To establish the first condition of verifier soundness, we prove that \( |Pr_Y^X(\tau(X,I)) - Pr_{\tilde{Y}}^X(\tau(\tilde{X},I))| \leq \varepsilon (|I|), \) i.e., \( \tilde{Y} \) can not distinguish \( \tilde{X} \) from any other \( X \). Note that the knowledge of a valid \( s_x \), such that \((g_y)^{s_x} = \ell \) is what distinguishes \( \tilde{X} \) from any other \( X \).

We show that if there exists a probabilistic polynomial time algorithm \( D \) that breaks the first condition of verifier soundness for A-CDLL, i.e., \( D \) can distinguish \( \tilde{X} \) from any other \( X \) on a given \( I \) \( |Pr_D^X(\tau(X,I)) - Pr_{\tilde{D}}^X(\tau(\tilde{X},I))| > \varepsilon (|I|) \), then we construct a probabilistic polynomial time algorithm \( \hat{A} \) that breaks the DDH intractability assumption as follows.

\( \hat{A}(g, R_1, R_2, \hat{K}) \)

- Randomly choose \( \hat{f} \), and let \( g_x = R_1, \ell = \hat{K}, g_y = \hat{K}^{\hat{f}}, \) and \( W = R_1^{\ell} \) (assuming \( s_x = 1/\hat{f} \), known to \( \hat{A} \)).
- Receive \( B_x, B_y, \ell, \hat{g} \) and \( n_1 \).
- Choose random \( u \). Send \( M = R_1^u \), and \( K = (\ell^u B_x^{n_1}{\hat{f}} B_y^{n_2}{\hat{f}}) \).
- Based on the output of \( D \), output 1, or 0.

In this construction, we assume that an honest party \( \tilde{Y} \) is one that knows \( s_y \) such that \( g^{s_y} = R_2 \). If \( D \) succeeds with non negligible probability in breaking A-CDLL without knowledge of \( s_y \), then \( \hat{A} \) will be able to successfully determine if \( \hat{K} \) is the correct DH key of
\( R_1 \) and \( R_2 \), hence violating the DDH assumption for \( g, R_1 \) and \( R_2 \). Clearly the construction \( \hat{A} \) is an efficient one, and is guaranteed to succeed with non negligible probability if adversary \( D \) succeeds in distinguishing \( \bar{X} \) from \( X \) with non negligible probability. The conversion of the DDH parameters to the assertion input involves setting \( g_x \) to \( R_1 \), \( \ell \) to \( \hat{K} \), \( g_y \) to \( \hat{K}^{\hat{f}} \), and \( W \) to \( R_1^{\hat{f}} \). We also assume that \( s_x = 1 / \hat{f} \). The test of the DH key is done using \( M \) and \( \ell^u \) used in deriving \( K \). Then if \( \hat{K} \) is the correct DH key, then the response \( M \) and \( K \) can be successfully verified, otherwise, it should fail.

Note that an honest prover will always send random values if its test on the values of \( \ell \) and \( \hat{g} \) fails. I.e., for the case where \( \bar{Y} \) fails to send correct values for these, \( \bar{X} \) send randoms for \( K \) and \( M \), i.e., behaves exactly as some dishonest strategy \( \bar{X} \). It then follows that \( D \) can not distinguish between \( \bar{X} \) and \( \bar{X} \) when it sends inconsistent values for \( \ell \) and \( \hat{g} \), because they are simply identical. That is why we considered in the above reduction the case where \( \bar{X} \)'s behavior is different than \( \bar{X} \), i.e., the case where the values for \( \ell \) and \( \hat{g} \) pass the verification at \( \bar{X} \).

Hence, we conclude that since there does not exists \( \hat{A} \) that breaks the DDH assumption in polynomial time with non negligible probability, then there does not exist \( D \) that breaks A-CDLL first condition of verifier soundness in polynomial time with non negligible probability.

\( D \) as \( \tilde{X} \): The second condition of verifier soundness trivially holds for A-CDLL, since an honest party \( \bar{Y} \) relies on its random coins only in formulating its message to \( \tilde{X} \). It then follows that it can not be distinguished from any other \( Y \).

\[ \textbf{Theorem 6.2} \quad \text{A-CDLL}(\bar{X}, \bar{Y}) \text{ is a proof system of membership in the common language DL2}-\hat{L}_{T_1}(\bar{X}, \bar{Y}). \]
Proof  For any $\hat{C}$, our proof considers two cases for $\hat{C}$. The first is the case where $I$ is in the common language of $\hat{C}$ and $\overline{X}$ or $\overline{Y}$, but $\hat{C}$ may exhibit arbitrary behavior. In that case, we show a simulation of $\overline{X}$ using the knowledge that $\hat{C}$ has of $I$, and a simulation of $\overline{Y}$ using the knowledge that $\hat{C}$ has of $I$. The second case, is when $I$ is not in the common language of $\hat{C}$ and $\overline{X}$ or $\overline{Y}$.

We start by the first case, i.e., when $I$ is in the common language of $\hat{C}$ and $\overline{X}$ or $\overline{Y}$. We start by showing a simulation for the honest party $\overline{X}$ done by simulator $M_{\hat{C}}(I)$, i.e., $M_{\hat{C}}(I)$ using the knowledge of $\hat{C}$ as follows:

Simulation $M_{\hat{C}}(I = \langle \ell, W_y, g_x, q, p \rangle)$

- Step 1: Let $\hat{C}$ send the first message.

- Step 2: Choose $\hat{r} \in \mathbb{U}[0,q-1]$, and compute $r = \text{rand}(\ell, \hat{g}, B, n)$, derive $M = g_x^r$ and $K = B_x^{n_1s_1}B_y^{n_2/s_2}$.

- Step 3: Send $M$, $n_2$ and $K$ to $\hat{C}$.

This is a polynomial time simulation on a given $I$ which will succeed in convincing any $\hat{C}$ that it is interacting with $\overline{X}$ on $I$ in the common language of both. This simulation will produce a probability ensemble $\{M_{\hat{C}}(I)\}_{I \in L_{\hat{C}}(\overline{X}, \hat{C})}$ that is computationally indistinguishable from the probability ensemble $\{(\overline{X}, \hat{C})(I)\}_{I \in L_{\hat{C}}(\overline{X}, \hat{C})}$.

On the other hand for strategy $\hat{C}$, and for any accepting interaction $[(\hat{C}, \overline{Y})(I)]$, $\overline{Y}$ uses only its random coins for deriving its message. It then follows that nothing can be leaked from $\overline{Y}$. Regardless of the state of knowledge of $\hat{C}$.
Next, we consider \( \hat{C} \) that has no knowledge of \( I \), i.e., \( I \) is not in the common language of \( \overline{X} \), or \( \overline{Y} \). \( \overline{Y} \) uses only its random coins, so no knowledge can be leaked from \( \overline{Y} \). We now consider the interaction of arbitrary \( \hat{C} \) and honest party \( \overline{X} \) and show a simulation that uses the strategy of \( \hat{C} \) to simulate \( \overline{X} \) as follows:

Simulation \( M_{\hat{C}}(I = \langle \ell, W, g_y, g_x, q, p \rangle) \)

- Step 1: Let \( \hat{C} \) send the first message.

- Step 2: Choose \( r \in [0, q-1] \), and compute \( r = rand(\ell, \hat{g}, B, n) \), and derive \( M = g_x^r \).

- Step 3: If \( b \) is known, derive \( K = \hat{g}^{n_1 b}W^{n_2 b} \). Send \( M \) and \( K \). Else, go to the next step.

- Step 5: Send \( M \) and random \( K \).

Based on the proven verifier soundness, \( \hat{C} \) can not distinguish this simulation from an honest interaction with \( \overline{X} \). Hence, no knowledge is leaked from \( \overline{X} \) to \( \hat{C} \).

\[\blacksquare\]

6.2 A-CRSAL Protocol for Identification based on Proofs on a Common RSA Language

This protocol is based on the hardness of inverting an RSA exponent \( e \). We assume that each of the parties are given either \( e \) or the inverse \( d \) of \( e \) such that \( ed = 1 \mod(\phi) \).

We denote the exponent of \( X \) by \( w_1 \) and the exponent of \( Y \) by \( W_2 \). Another private input to each is the flag, which defines the protocol of the party. The protocol is based on the intractability of the RSA assumption assuming the modulus \( m \) is sufficiently large and randomly generated. All computations are assumed mod \( m \). The protocol is a proof of membership in the common language \( RSA1-L_{T_\tau}(\overline{X}, \overline{Y}) \) specified in Table 4.6.
The protocol is given in Table 6.2. We assume that an honest party $\mathcal{Y}$ Checks if $(\hat{W} = \hat{M}^{w_2})$, or if $(\hat{M} = \hat{W}^{w_2})$, if both are false, $\mathcal{Y}$ send random values for $C, B, r'_2$, since $\tau(Y, I)$ is false, for $I = (\hat{W}, \hat{M}, m)$, i.e., $I$ is not in a common language of any $X$ and $\mathcal{Y}$. Similarly, we assume that an honest party $\mathcal{X}$ Checks if $(\hat{W} = \hat{M}^{w_1})$, or if $(\hat{M} = \hat{W}^{w_1})$, if both are false, $\mathcal{X}$ sends random values for $K$ and $R$, since $\tau(X, I)$ is false, for $I = (\hat{W}, \hat{M}, m)$, i.e., $I$ is not in a common language of $\mathcal{X}$ and any $Y$.

The protocol uses two messages, one sent from $\mathcal{Y}$ to $\mathcal{X}$ containing three values $C, B$ and $r'_2$, and the other sent from $\mathcal{X}$ to $\mathcal{Y}$ containing two values $K$ and $R$. $\mathcal{Y}$ randomly chooses $r_1$ and $r_2$, and then uses its knowledge of RSA exponent $w_2$ to derive the values $C, B$ and $r'_2$ based on its flag $F_v$. This flag is given to $\mathcal{Y}$ along with its exponent $w_2$, and it defines the way $\mathcal{Y}$ formulates its message as well as the check it performs. Upon receiving $C, B$ and $r'_2$, and based on its flag $F_p$, $\mathcal{X}$ chooses a random $r$, and derives $l$ using its RSA exponent $w_1$ as well as the values sent by $\mathcal{Y}$. $\mathcal{X}$ then uses the RSA exponent $t$ along with $r, l$ and its exponent $w_1$ to compute $K$ and $R$. Note that $t$ is an RSA exponent whose inverse is not known to $\mathcal{X}$ or $\mathcal{Y}$.

**Theorem 6.3**  
$A$-$\text{CRSAL}(\mathcal{X}, \mathcal{Y})$ is a proof system of membership in a common RSA language based on the intractability of the RSA problem.

**Proof**  
Assume negligible function $\varepsilon : N \to [0, 1] : \varepsilon(n) = 2^{-n}$, We first prove Completeness followed by Prover and Verifier Soundness.
<table>
<thead>
<tr>
<th>Party ((X(w_1, F_p))) ((\langle \tilde{W}, \tilde{M}, m \rangle, L = 1, t))</th>
<th>Party ((Y(w_2, F_v))) (n_1, r_1, r_2 \in u[0, m - 1])</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF (F_v = 1) (C = n_1 r_1 + (r_2)^{w_2}) (B = (r_1)^{w_2}) (r_2' = r_2) ELSE (C = n_1 (r_1)^{w_2} + r_2) (B = r_1) (r_2' = (r_2)^{w_2})</td>
<td></td>
</tr>
</tbody>
</table>
| \(r \in u[0, m - 1]\) IF \(F_p = 1\) \(l \equiv (r_2')^{w_1}\) Check: \(((C - l)n_1^{-1})^{w_1} \equiv B\) If False, send random \(R\) and \(K\) Else \(R \equiv (lr)\) \(K \equiv r^{w_1 t}\) ELSE \(l \equiv ((C - n_1 B)^{w_1})\) Check \(l^{w_1} \equiv r_2'\) If False, send random \(R\) and \(K\) Else \(R \equiv (lr^{w_1})\) \(K \equiv r^t\) | \(R, K\)  
\[R, K \rightarrow\] IF \(F_v = 1\) Check: \((Rr_2^{-w_2})^{wlf} \equiv K\) ELSE Check: \((Rr_2^{-1})^t \equiv K^{w_2}\) |

Table 6.2: A-CRSAL Protocol for Identification based on Proofs on Common RSA Language for non Identical Prover Strategies
Completeness  To prove completeness, we show that an honest prover that knows \( w_1 \) such that \( w_1 = w_2 \) (\( F_p = F_v \)) or \((w_1w_2 \equiv 1 (mod \phi(m)) \) (\( F_p \neq F_v \)) satisfies the verification condition with probability 1.

- **Case 1** (\( F_p = F_v = 1 \)) (i.e., \( w_1 = w_2 \)):

  \[
  (Rr_2^{-w_2})^{w_2} \equiv ((lr)(r_2^{-w_2}))^{w_2} (mod m)
  \]
  \[
  \equiv ((r_2^{w_1}r)(r_2^{-w_2}))^{w_2} (mod m)
  \]
  \[
  \equiv (r_2^{w_1}r r_2^{-w_2})^{w_2} (mod m) \), since \( w_1 = w_2 
  \]
  \[
  \equiv (r^{w_2})(mod m)
  \]
  \[
  \equiv K(mod m) \), since \( w_1 = w_2 
  \]

- **Case 2** (\( F_p = F_v = 0 \)) (i.e., \( w_1 = w_2 \)):

  \[
  (Rr_2^{-1})^t \equiv ((lr^{w_1})(r_2^{-1}))^t (mod m)
  \]
  \[
  \equiv (((C - n_1B^{w_1})r^{w_1})(r_2^{-1}))^t (mod m)
  \]
  \[
  \equiv (((n_1 r_1^{w_2} + r_2 - n_1 (r_1)^{w_1})r^{w_1})(r_2^{-1}))^t (mod m)
  \]
  \[
  \equiv (r_2^{w_1}r_2^{-1})^t (mod m)
  \]
  \[
  \equiv r^{w_1} (mod m)
  \]
  \[
  \equiv K^{w_1} (mod m)
  \]
  \[
  \equiv K^{w_2} (mod m) \), since \( w_1 = w_2 
  \]

- **Case 3** (\( F_p = 1 \land F_v = 0 \)) (i.e., \( w_1 \equiv 1 (mod \phi(m)) \)):

  \[
  (Rr_2^{-1})^t \equiv ((lr)(r_2^{-1}))^t (mod m)
  \]
  \[
  \equiv ((r_2^{w_1}r)(r_2^{-1}))^t (mod m)
  \]
  \[
  \equiv (((r_2^{w_2})r_2^{-1}))^t (mod m)
  \]
\equiv (r_2 r^{-1})^t (mod m)
\equiv r^t (mod m)
\equiv (r^{w_1})^{w_2} (mod m), \text{ since } (w_1 w_2 \equiv 1 (mod\phi(m))
\equiv K^{w_2} (mod m)

- **Case 4** ($F_p = 0 \land F_v = 1$ (i.e., $(w_1 w_2 \equiv 1 (mod\phi(m))$ )):

$$(R r_2^{-w_2})^{w_2 t} \equiv ((lr^{w_1})(r_2^{-w_2}))^{w_2 t} (mod m)$$
\equiv (((C - n_1 B r^{w_1})(r_2^{-w_2}))^{w_2 t} (mod m)
\equiv (((C - n_1 (r_1 w_2)^{w_1})(r_2^{-w_2}))^{w_2 t} (mod m)
\equiv (((C - n_1 r_1)^{w_1})(r_2^{-w_2}))^{w_2 t} (mod m), \text{ since } (w_1 w_2 \equiv 1 (mod\phi(m))
\equiv ((r_2^{-w_2} r^{w_1})(r_2^{-w_2}))^{w_2 t} (mod m)
\equiv (r^{w_1 w_2 t}) (mod m)
\equiv (r^t) (mod m)
\equiv K (mod m)

**Prover Soundness:** To establish Prover soundness, assume $\tilde{X}$ does not know $w_1$ such that $w_1 = w_2$ or $(w_1 w_2 \equiv 1 (mod\phi(m)))$, also assume factoring $m$ is hard, hence, the RSA assumption holds. We show that if $\bar{Y}$ follows the protocol and $\tilde{X}$ performs arbitrary polynomial time computations, $\bar{Y}$ accepts the proof of $\tilde{X}$ as valid with probability bounded by $\varepsilon(|I|)$

**Case 1:** $F_v = 1$ $\tilde{X}$ can cheat by guessing a correct value of $r_1$ from $B$ and then sending $R = ((C - n_1 r_1)(r_1)(mod m)$, and $K = B^t$. Nonetheless, the probability of success of this strategy is only $\varepsilon(|I|)$.
To prove that the success probability of $\tilde{X}$ is bounded by $\varepsilon(|I|)$, we show that if there exists a probabilistic polynomial time algorithm $A$ that breaks prover soundness for case 1 verifier, i.e., convinces $Y$ that $I$ is in a common language with non negligible probability, then we construct a probabilistic polynomial time algorithm $\hat{A}$ that breaks the RSA intractability assumption as follows:

Assume that $\tilde{X}$ does not know a value for $r_1$, hence it can not derive $r_2w_2$, and that it can successfully pass the verification with non negligible probability. We then show that an Oracle $RSA-1(e, I)$ that uses $\tilde{X}$ can be used for for solving the RSA problem (i.e., outputs $s$ such that $s^e = I$), hence contradicting the assumed intractability.

$\hat{A}(e, \hat{W})$

- **Step 1:** Set $w_2 = e$ and follow $Y$ protocol in generating values of $r_1$ and $r_2$, and $n_1$, sending $C, B$ and $r'_2$ to $A$ and receiving $R$ and $K$. Repeat Step1 if the verification did not succeed. If succeeded, derive $a = Rr_2^{-w_2}$, reset $\tilde{X}$ to the same randomness on which $A$ succeeded and move on to the next step.

- **Step 2:** Generate $n_1, C$ and $r'_2$ randomly, and let $B = \hat{W}$. send $n_1, B, C$ and $r'_2$ to $\tilde{X}$.

- **Step 3:** Receive $R$ and $K$ from $A$, and compute $\hat{M} = (C - Ra^{-1})n_1^{-1}$. If $s^e = I$ return $\hat{M}$, otherwise reset $A$ and go to Step2.

Notice that $A$ can not perform the checks that $X$ does, i.e., it has no way of determining if the verifier is following its protocol. If $A$ succeeds with non negligible probability, then Oracle $\hat{A}$ succeeds with non negligible probability to compute $\hat{M}$ such that $\hat{M}^e = \hat{W}$. Also, notice that $\hat{M}$ would be the value of $r_1$ which we initially assumed is not known by $\tilde{X}$. (a contradiction).
Hence, we conclude that since there does not exist \( \hat{A} \) that breaks the RSA assumption in polynomial time with non negligible probability, then there does not exist \( A \) that breaks prover soundness for case 1 in polynomial time with non negligible probability.

**Case 2:** \( F_Y = 0 \) \( \tilde{X} \) can cheat by guessing a correct value of \( r_2 \) from \( r' \) and then sending 
\[
R = r_2((C - r_2)n^{-1})(mod\ m), \text{ and } K = B'.
\]
Similar to Case 1, the success probability of this strategy is only \( \epsilon(|I|) \).

To prove that the success probability of \( \tilde{X} \) is bounded by \( \epsilon(|I|) \), assume that \( \tilde{X} \) does not know a value for \( r_2 \), and that it can successfully pass the verification with non negligible probability, then we show that if there exists a probabilistic polynomial time algorithm \( A \) that breaks prover soundness for A-CRSAL for case 2 verifier, i.e., convinces \( Y \) that \( I \) is in a common language with non negligible probability, then we construct a probabilistic polynomial time algorithm \( \hat{A} \) that breaks the RSA intractability assumptions follows:

**Oracle** \( \hat{A}(e, \hat{W}) \)

- **Step 1:** Set \( w_2 = e \) and follow \( Y \) protocol in generating values of \( r_1 \) and \( r_2 \), and \( n_1 \), sending \( C, B \) and \( r'_2 \) to \( \tilde{P} \) and receiving \( R \) and \( K \). Repeat Step1 if the verification did not succeed. If succeeded, derive \( a = Rr_2^{-1} \), reset \( A \) to the same randomness on which \( A \) succeeded and move on to the next step.

- **Step 2:** Generate randoms \( n_1, C \) and \( B \), and let \( r'_2 = I \). send \( n_1, B, C \) and \( r'_2 \) to \( A \).

- **Step 3:** Receive \( R \) and \( K \) from \( A \). and compute \( \hat{M} = Ra^{-1} \). If \( \hat{M}^e = \hat{W} \) return \( \hat{M} \), otherwise reset \( A \) and go to Step2.

If \( A \) succeeds with non negligible probability, then RSA solver \( \hat{A} \) succeeds with non negligible probability to compute \( \hat{M} \) such that \( \hat{M}^e = \hat{W} \). Also, notice that \( \hat{M} \) would be the value of \( r_2 \) which we initially assumed is not known by \( A \). (a contradiction).
Hence, we conclude that since there does not exist \( A \) that breaks the RSA assumption in polynomial time with non negligible probability, then there does not exist \( A \) that breaks prover soundness for case 2 verifier in polynomial time with non negligible probability.

**Verifier Soundness:** To prove verifier soundness, we shall consider all polynomial time strategies \( D \) and black box interactions with \( X \) and \( \tilde{X} \) or \( Y \) and \( \tilde{Y} \). In the distinguishing experiment of \( X \), \( D \) impersonates \( Y \) to \( X \), i.e., it can be considered as \( \tilde{Y} \), since it does not have knowledge of the correct RSA exponent. On the other hand, in the distinguishing experiment involving \( Y \), \( D \) impersonates \( X \) to \( Y \), i.e., it can be considered as \( \tilde{X} \), since it does not have knowledge of the discrete log of the RSA exponent.

**\( D \) as \( \tilde{Y} \):** Regardless of whether or not \( Y \) is honest, an honest prover performs a check on the values \( n, C, B \) and \( r_2' \) sent by the verifier, and if the verification fails, \( X \) sends random values for \( R \) and \( K \), otherwise, it follows the protocol and responds accordingly. As proved in soundness, \( \tilde{X} \) can not succeed to fool \( Y \) except with negligible probability, i.e., any strategy that \( \tilde{X} \) deploys is no better than a random guess, and since \( X \) employs a random guess strategy for \( K \) and \( R \), then both will be accepted with similar probabilities at any verifier. This assumes that honest verifiers are able to pass the check performed at \( X \) with probability 1, and that dishonest verifiers can only pass this check with negligible probability. We show that if this assumption is not true, i.e., there exists an efficient non-uniform probabilistic dishonest verifier that can pass the check at \( P \) with non-negligible probability, then an RSA solver can be derived based on its strategy, hence violating the RSA assumption.

Assume \( I \) is not in the common language of \( X \) and \( \tilde{Y} \), or that \( \tilde{Y} \) knows a valid \( w_2 \) but does not follow the protocol (hence, \( \tilde{Y} \) may not end up using \( w_2 \) as specified by the protocol, for example, to exploit whether or not \( w_1 = w_2 \)), and assume that \( \tilde{Y} \) passes the check at \( X \)
with non negligible probability. We show that the strategy of $\tilde{Y}$ can be used to construct RSA solver as follows:

$\tilde{Y}$ can pass $X$’s check: $l^{w_1} \equiv r'_2 (\text{mod } m)$ if $F_P = 0$ and $((C-l)n_1^{-1})^{w_1} \equiv B (\text{mod } m)$ if $F_P = 1$, with non negligible probability. We show the following efficient RSA solver that simulates $\tilde{Y}$ and assumes that $\tilde{Y}$ can pass the verification with non negligible probability.

The solver uses the oracle $H(e,\text{FLAG})$ which uses the RSA exponent $e$ in the same way $w_1$ is used to check the values sent by $Y$ at $X$. I.e., when queried with parameters $C$, $r'_2$, $n_1$ and $B$, the oracle $H(e,\text{FLAG})$ performs the check $l^{w_1} \equiv r'_2 (\text{mod } m)$ if $\text{FLAG}=0$, or $((C-l)n_1^{-1})^{w_1} \equiv B (\text{mod } m)$ if $\text{FLAG}=1$, and outputs 1 if the check succeeds and 0 otherwise. The use of this checker oracle is to guarantee that the solver has no access to $X$ or any of the valid values of $w_1$ and $w_2$, but when given the auxiliary knowledge of whether the check fails or succeeds, it can efficiently obtain $s$ such that $s^e = I$, hence contradicting the intractability assumption of the RSA problem.

$\text{RSA-VSolver}(H(e,\text{FLAG}), I, F_P)$

- Step 1: Generate $n_1, C$ If $F_P = 1$ then generate $r'_2$ randomly, and let $B = I$, otherwise, generate $B$ randomly and let $r'_2 = I$. Query the oracle $H(e,F_P)$ with $B$, $C$ and $r'_2$. If $H(e,F_P)$ outputs 1, then the check succeeded, move on to Step 2. Otherwise, repeat Step 1.

- Step 2: Generate a new value for $n_1$ and $C$, and fix all other values. Send all the values over to $H(e,F_P)$. Repeat Step2 until $H(e,F_P)$ outputs 1 for some $n_1$ that is different than the $n_1$ used in Step1. We shall denote the $n_1$ chosen in step 1 as $st1.n_1$, and the $n_1$ in step 2 as $st2.n_1$, similarly denote the value of $C$ chosen in step 1 as $st1.C$, and the $C$ chosen in step 2 as $st2.C$. Upon success, move on to the next step.
\* Step 2: Return \( s = (st1.C - st2.C)(st1.n_1 - st2.n_1)^{-1} \).

**D as \( \tilde{X} \):** The second verifier soundness condition holds for A-CRSAL, since the message sent by \( \tilde{Y} \) can be efficiently constructed using \( I \) only, i.e., without knowledge of the RSA exponent as follows: Choose random exponent \( u, v \), compute \( \tilde{W}^u \) and \( \tilde{M}^u \), repeat for \( v \), then use the values for \( r_1 \) and \( r_2 \) in \( C \) and \( B \) for both flag cases.

\[ \square \]

**Theorem 6.4** A-CRSAL(\( \tilde{X}, \tilde{Y} \)) is a zero knowledge proof system of membership in a common RSA language based on the intractability of the RSA problem.

**Proof** For any \( \hat{C} \), our proof considers two cases for \( \hat{C} \). The first is the case where \( I \) is in the common language of \( \hat{C} \) and \( \tilde{X} \) or \( \tilde{Y} \), but \( \hat{C} \) may exhibit arbitrary behavior. In that case, we show a simulation of \( \tilde{X} \) using the knowledge that \( \hat{C} \) has of \( I \), and a simulation of \( \tilde{Y} \) using the knowledge that \( \hat{C} \) has of \( I \). The second case, is when \( I \) is not in the common language of \( \hat{C} \) and \( \tilde{X} \) or \( \tilde{Y} \).

We start by the first case, i.e., when \( I \) is in the common language of \( \hat{C} \) and \( \tilde{X} \) or \( \tilde{Y} \). We start by showing a simulation for the honest party \( \tilde{X} \) done by simulator \( M_{\hat{C}}(I) \), i.e., \( M_{\hat{C}}(I) \) using the knowledge of \( \hat{C} \) as follows:

Simulation \( M_{\hat{C}}(I = \langle \tilde{W}, \tilde{M}, m \rangle) \)

\* Step 1: Receive the first message from \( \hat{C} \).

\* Step 2: If \( F_v = 1 \) follow the protocol for \( \tilde{X} \) when \( F_p = 1 \), else, follow the other protocol.

This is a polynomial time simulation on a given \( I \) which will succeed in convincing \( \hat{C} \) that it is interacting with \( \tilde{X} \). This simulation will produce a probability ensemble

---

136
\( \{ M_\hat{C}(I) \}_{I \in L_{\tau}(X,Y)} \) that is computationally indistinguishable from the probability ensemble
\( \{(X, \hat{C})(I)\}_{I \in L_{\tau}(X,Y)} \).

Similarly, the message of \( \overline{Y} \) to \( X \) can be easily simulated using the vector \( I \) as follows:
Choose random exponent \( u,v \), compute \( \hat{W}^u \) and \( \hat{M}^v \), repeat for \( v \), then use the values for \( r_1 \) and \( r_2 \) in \( C \) and \( B \) for both flag cases.

Next, we consider \( \hat{C} \) that has no knowledge of \( I \), i.e., \( I \) is not in the common language of \( X \), or \( \overline{Y} \). We first consider the interaction of arbitrary \( \hat{C} \) and honest party \( X \) and show a simulation that uses the strategy of \( \hat{C} \) to simulate \( X \) as follows:

Simulation \( M_{\hat{C}}(I = \langle \hat{W}, \hat{M}, m \rangle) \)

- Step 1: Let \( \hat{C} \) send the first message.

- Step 2: choose random \( w \), and let \( r = \hat{M}^w \), also derive \( \hat{W}^w \). If \( l \) is known, follow the protocol for \( X \) to formulate the messages using \( r \), \( \hat{W}^w \) and \( l \). Otherwise, send randoms for \( R \) and \( K \).

Based on the proven verifier soundness, \( \hat{C} \) can not distinguish this simulation from an honest interaction with \( X \). Hence, no knowledge is leaked from \( X \) to \( \hat{C} \).

Similarly, the message of \( \overline{Y} \) to \( \hat{C} \) can be easily simulated using the vector \( I \) as follows:
Choose random exponent \( u,v \), compute \( \hat{W}^u \) and \( \hat{M}^v \), repeat for \( v \), then use the values for \( r_1 \) and \( r_2 \) in \( C \) and \( B \) for both flag cases.

\[ \square \]

### 6.3 A-CLHG Protocol for Identification based on a Common Graph Hamiltonian Language

The A-CLHG protocol in Table 6.3 assumes that \( \overline{X} \) knows a cycle \( C_x \) of \( G \) and \( \overline{Y} \) knows a cycle \( C_y \) of \( G \), such that both cycles are Hamiltonian cycles, and \( C_x = \eta C_y \) such that
G_x = \eta G \text{ and } G_y = \eta^{-1} G. \text{ This protocol can also be used for the case when } C_x = C_y \text{ and } C_x \text{ is a cycle in both } G \text{ and } G_x \text{ or } G \text{ and } G_y. \text{ To simplify our discussions, we focus on the case where } C_x \text{ is different than } C_y. \text{ The interaction starts on common input } \langle G, G_x, G_y \rangle, \text{ and the first message is sent from } \bar{Y} \text{ and it contains a permuted cycle } B_1 \text{ for } C_y \text{ of } Y. \bar{X} \text{ then uses its cycle } C_x \text{ to derive } \eta_2 \text{ from } B_1 \text{ and check that } B_1 \text{ is a cycle in } \eta_2 G_x, \text{ which is equivalent to } \eta_1 G, \text{ and } \eta_2 G, \text{ which is equivalent to } \eta_1 G_y. \text{ If } B_1 \text{ did not pass both check conditions, } X \text{ sends a random value for } R \text{ and } K. \text{ Otherwise it permutes its cycle } C_x \text{ using } \eta_3 \text{ and uses the incidence matrix of } R = \eta_2 G_x \text{ XORed with the incidence matrix of } \eta_3 G \text{ to create } M, \text{ and it sends also } K = \eta_3 C_x. Y \text{ then performs the verification as indicated in Table 6.3. The protocol is a proof of membership in the common language } HML2-\hat{L}_{\tau_T}(\bar{X}, \bar{Y}) \text{ specified in Table 4.9. The A-CLHG protocol and security analysis assumes the hardness of finding Hamiltonian cycles in graphs, as well as the the hardness of the decidability of the graphs isomorphism problem. I.e., given any two graphs, there is no efficient algorithm that can decide if these graphs are isomorphic or not.}

The protocol involves operations on the incidence matrix, which is simply a matrix that completely specifies the graph as follows: If graph } G \text{ has } n \text{ vertices, then } \phi(G) \text{ is an nxn matrix, such that the } (i, j) \text{’th entry is 1 if there is an edge in } G \text{ from vertex } i \text{ to vertex } j, \text{ and is 0 otherwise. Note that if we have a permutation } \eta \text{ on } n \text{ elements and apply it to } G, \text{ we can obtain the incidence matrix of } \eta(G) \text{ easily from } \phi(G) \text{ by permuting the rows and columns of } \phi(G) \text{ according to } \eta. \text{ The resulting matrix is called } \eta(\phi(G)). \text{ Note that a Hamiltonian cycle } C \text{ in } G \text{ can be specified by pointing out a set of } n \text{ entries in } \phi(G) \text{ such that all the entries are 1, and such that they specify a cycle that visits every node exactly once. XORing of two incidence matrices involves XORing of their individual corresponding entries.}
\( I = (G, G_x, G_y) \)
\( C_x, C_y \) cycles of \( G \)
\( (C_x = \eta C_y) \)
\( (G_x = \eta G), (C_x \text{ cycle of } G_x) \)
\( (G_y = \eta^{-1} G), (C_y \text{ cycle of } G_y) \)

Choose \( \eta_1 \)
\( B_1 = \eta_1(C_y) \)

Derive \( \eta_2 \) s.t.:
\( \eta_2(C_x) = B_1 \) and
Check:
\( B_1 \) cycle in \( \eta_2(G_x) \)
and \( \eta_2(G) \)
If not, send
randoms for \( R, K \)
Else:
Choose random \( \hat{\eta} \)
\( \eta_3 = \text{rand}(\hat{\eta}, B_1) \)
\( R = \eta_2 G_x \)
\( K = \eta_3(C_x) \)
\( M = R \text{ XOR } \eta_3(G) \)

Derive \( \eta_4 \) s.t.
\( \eta_4(C_y) = K \)
Validate:
\( \eta_1(G) = \eta_4(G_y) \text{ XOR } M \)

Table 6.3: A-CLHG Protocol for Identification based on Common Graph Hamiltonian Language for non Identical Prover Strategies
**Theorem 6.5**  
A-CLHG(\(X, Y\)) is a proof of membership in a common Graph Hamiltonian language assuming the intractability of finding Hamiltonian Cycles and the intractability of the graphs isomorphism decidability problem.

**Proof**  
Assume negligible function \(\epsilon : N \rightarrow [0, 1] : \epsilon(n) = 2^{-n}\), We show that A-CLHG satisfies Completeness, Prover and Verifier Soundness in that order.

**Completeness**  
To prove completeness, we show that an honest prover that knows a Hamiltonian Cycle \(C_X\) in Graph \(G_x\) and \(G\) such that \(C_x = \eta C_y\) for any \(\eta\) such that \(G_y = \eta^{-1} G\), succeeds in satisfying the verification condition with probability 1.

\[
Y \text{ derives } \eta_4 G_y \\
= (\eta_3 \eta) G_y \\
= (\eta_3 \eta) \eta^{-1} G \\
= \eta_3 G
\]

\[
\eta_3 G \text{ XOR } M = \eta_1 G \\
= \eta_2 \eta G \\
= \eta_2 G_x
\]

Note that \(Y\) always passes the check at \(X\), since \(\eta_1(C_y) = \eta_1 \eta^{-1}(C_x)\), i.e., \(X\) derives \(\eta_2 = \eta_1 \eta^{-1}\), so \(X\) computes \(\eta_2(G_x) = \eta_1(G)\), and for \(Y\), \(B_1\) is guaranteed to be a cycle for \(\eta_1(G)\).
**Prover Soundness:** To establish Prover soundness, assume $\tilde{X}$ does not know the Hamiltonian Cycle $C_x$ of Graph $G_x$, nor does it know $C_y$ of graph $G_y$. Assume that $\tilde{X}$ cheats by guessing $(\eta_1)$ correctly, then it can successfully formulate $R$. Nonetheless, if $\tilde{X}$ can guess $(\eta_1)$ correctly with non negligible probability, then $\tilde{X}$ can successfully derive $C_y$ with non negligible probability, i.e., $\tilde{X}$ knows $C_y$, which contradicts the initial assumption that $\tilde{X}$ does not know $C_y$. Therefore, it must be that $\tilde{X}$ can not successfully guess $\eta_1$ except with negligible probability.

We show that if there exists a probabilistic polynomial time algorithm $A$ that breaks prover soundness for A-CLHG, i.e., convinces $Y$ that $I$ is in a common language with non negligible probability, then it must be that $A$ is an efficient graph isomorphism solver, hence we use $A$ to construct a probabilistic polynomial time algorithm $\hat{A}$ that breaks the Graph Isomorphism decidability problem. $\hat{A}$ takes in two graphs $G_i$ and $G_j$ as well as a cycle $c_j$ in $G_j$. $\hat{A}$ outputs 1 if $G_i$ is isomorphic to $G_j$ and outputs 0 otherwise. Under the intractability assumptions for the Graph isomorphism decidability, an efficient solver exists only if the isomorphism between the two given graphs is known. Let $G$ be $G_i$, then the construction for the solver will then be as follows:

$\hat{A}(c_j, G_i, G_j)$

- Choose random $\eta_1$, and send $B_1 = \eta_1 c_j$.
- Receive $R$ and $K$.
- Use $\eta_1 G_j$ XORed with $R$ to derive $val = \eta_3 G$.
- If $K$ is a cycle in $val$, return 1, else, return 0.

If $A$ succeeds with non negligible probability in breaking A-CLHG prover soundness, then $\hat{A}$ will be able to successfully determine if $G_i$ and $G_j$ are isomorphic. Note that honest
strategy $\overline{X}$ can not be fooled by this strategy because of the check it performs, which $\overline{X}$ can not do as it does not know a cycle of $G$. The construction $\hat{A}$ is an efficient one, and is guaranteed to succeed with non negligible probability if adversary $A$ succeeds in impersonating $\overline{X}$ at $\overline{Y}$ with non negligible probability. If $G_i$ and $G_j$ are isomorphic, then there exists an isomorphism $\pi$ such that $G_j = \pi G_i$, and if a cycle $c_j$ is known in $G_j$, then it is not possible to get the cycle of $G_i$ without knowing $\pi$. $\hat{A}$ sets $G$ to $G_i$, the graph for which it does not know a Hamiltonian cycle. Then $\hat{A}$ sends the permuted cycle $c_j$ of $G_j$ and uses the permuted graph $G_j$ to derive $val$ and check that $K$ is a cycle of $val$. If $G_i$ and $G_j$ are isomorphic, then $K$ will be a cycle of $val$, otherwise, it will not, as the value that is forming $R$ is not the same as $\eta_1 G_j$.

Hence, we conclude that since there does not exist $\hat{A}$ that breaks the graph isomorphism decidability in polynomial time with non negligible probability, then there does not exist $A$ that breaks A-CLHG soundness in polynomial time with non negligible probability.

**Verifier Soundness:** To prove verifier soundness, we shall consider all polynomial time strategies $D$ and black box interactions with $\overline{X}$ and $\overline{X}$ or $\overline{Y}$ and $\overline{Y}$. In the distinguishing experiment of $\overline{X}$, $D$ impersonates $\overline{Y}$ to $\overline{X}$, i.e., it can be considered as $\overline{Y}$, since it does not have knowledge of a Hamiltonian cycle $C_x$ or $C_y$ of $G$. On the other hand, in the distinguishing experiment involving $\overline{Y}$, $D$ impersonates $\overline{X}$ to $\overline{Y}$, i.e., it can be considered as $\overline{X}$, since it does not have knowledge of a Hamiltonian cycle $C_x$ or $C_y$ of $G$.

**$D$ as $\overline{Y}$:** We show that if there exists a probabilistic polynomial time algorithm $D$ that breaks first condition of verifier soundness for A-CLHG, i.e., $D$ can distinguish $\overline{X}$ from any other $X$ on a given $I$ ($|Pr_D^{\overline{X}}(\tau(X,I)) - Pr_D^{\overline{X}}(\tau(\overline{X},I))| > \varepsilon (|I|))$, then we construct a
probabilistic polynomial time algorithm $\hat{A}$ that breaks the graph Isomorphism decidability assumption as follows:

$\hat{A}(c_j, G_i, G_j)$

- Let $G = G_i$.
- $D$ sends $B_1$.
- Use $c_j$ to derive $\eta_2$ and then derive $R$ and $K$ and send it to $D$.
- Based on the output of $D$, return 1, i.e., $G_i$ isomorphic to $G_j$ if $A$ outputs 1, and return 0 otherwise.

If $D$ succeeds with non negligible probability in breaking the verifier soundness of $A$-CLHG, then $\hat{A}$ will be able to successfully determine if $G_i$ and $G_j$ are isomorphic. The construction $\hat{A}$ is an efficient one, and is guaranteed to succeed with non negligible probability if adversary $D$ succeeds in impersonating $Y$ with non negligible probability, i.e., if it can distinguish $X$ from $X$ with non negligible probability. If $G_i$ and $G_j$ are isomorphic, then there exists an isomorphism $\pi$ such that $G_j=\pi G_i$, and if a cycle $c_j$ is known in $G_j$, then $\hat{A}$ derives $\eta_2$ as the permutation, and uses it to derive $R = \eta_2 G_j$. Then if $D$ can successfully distinguish $\overline{X}$ from $X$ with non negligible probability, then its output can be used to decide the isomorphism of $G_j$ and $G_j$.

Note that an honest prover will always send random $R$ and $K$ if its test on the values $B_1$ and $G_1$ fails. I.e., for the case where $D$ fails to send correct values for these, $\overline{X}$ does not reply with correctly computed $R$, i.e., behaves exactly as some dishonest strategy $\tilde{X}$. It then follows that $D$ can not distinguish between $\overline{X}$ and $\tilde{X}$ when it send a value for $B_1$ that does not pass its check, specifically the second part of the check. That is why we considered in
the above reduction the case where $\overline{X}$'s behavior is different than $\overline{X}$, i.e., the case where the value for $B_1$ passes the check at $\overline{X}$.

Hence, we conclude that since there does not exists $\hat{A}$ that breaks the graph isomorphism decidability in polynomial time with non negligible probability, then there does not exist $D$ that breaks the first condition of A-CLHG verifier soundness in polynomial time with non negligible probability.

$D$ as $\overline{Y}$: For the second condition of verifier soundness, it suffices to note that any $Y$ can send random string for $B_1$ and with high probability, it may constitute a valid isomorphism on a cycle of $G$ (Note that $Y$ may not know a valid cycle of $G$ but can still send a valid value for $B_1$). Hence, the message of $Y$ can not be distinguishable from $\overline{Y}$, especially that even if $D$ uses $\overline{X}$ to detect a difference in the reply, it will still not be able to distinguish, as the reply of $\overline{X}$ can not be distinguished from the reply of $\overline{X}$ as proved for the first condition of verifier soundness. Also the values of the message sent by $\overline{Y}$ and $Y$ are indistinguishable, based on the hardness of finding the isomorphism, and also the hardness of finding a Hamiltonian cycle. Hence, $D$ can not distinguish $\overline{Y}$ from any other $Y$ on a given $I$ except with negligible probability. I.e., $|Pr_Y(D(\tau(Y,I))) - Pr_{\overline{Y}}(\tau(\overline{Y},I))| \leq \varepsilon (|I|)$.

\[\blacksquare\]

**Theorem 6.6** A-CLHG($\overline{X}, \overline{Y}$) is a zero knowledge proof of membership in a common Graph Hamiltonian language assuming the intractability of finding Hamiltonian Cycles and the intractability of the graphs isomorphism decidability problem.

**Proof** For any $\hat{C}$, our proof considers two cases for $\hat{C}$. The first is the case where $I$ is in the common language of $\hat{C}$ and $\overline{X}$ or $\overline{Y}$, but $\hat{C}$ may exhibit arbitrary behavior. In that case,
we show a simulation of $\overline{X}$ using the knowledge that $\hat{C}$ has of $I$, and a simulation of $\overline{Y}$ using the knowledge that $\hat{C}$ has of $I$. The second case, is when $I$ is not in the common language of $\hat{C}$ and $\overline{X}$ or $\overline{Y}$.

We start by the first case, i.e., when $I$ is in the common language of $\hat{C}$ and $\overline{X}$ or $\overline{Y}$. We show a simulation for the honest party $\overline{X}$ done by simulator $M_{\hat{C}}(I)$, i.e., $M_{\hat{C}}(I)$ using the knowledge of $\hat{C}$ as follows:

Simulation $M_{\hat{C}}(I = \langle G, G_x, G_y \rangle)$

- Step 1: Let $\hat{C}$ send the first message: $B_1$.

- Step 2: Use $C_y$, the knowledge of $\hat{C}$, to derive $\eta_2$, then compute $R = \eta_2 G, K = \eta_3 C_y$ for random $\eta_3$. Then derive $M = R \text{ XOR } \eta_3 G_y$.

- Step 3: Send $M, K$ to $\hat{C}$.

This is a polynomial time simulation on a given $I$ which will succeed in convincing any $\hat{C}$ that it is interacting with $\overline{X}$ on $I$ in the common language of both, since it can be successfully verified at $\hat{C}$. This is because $M_{\hat{C}}$ uses $C_y$ to derive $\eta_2$, hence, $\eta_2 = \eta_1$. Same applies for $\eta_3$ which will be equal to $\eta_4$. This simulation will produce a probability ensemble $\{M_{\hat{C}}(I)\}_{I \in L_{\hat{C}}(X, \hat{C})}$ that is computationally indistinguishable from the probability ensemble $\{\overline{(X, \hat{C})}(I)\}_{I \in L_{\hat{C}}(X, \hat{C})}$.

On the other hand for any party $\hat{C}$, and for any accepting interaction $[(X, \overline{Y})(I)]$, the message of $\overline{Y}$ can be easily simulated using the cycle of $\hat{C}$, i.e., $C_x$. 

145
Next, we consider \( \hat{C} \) that has no knowledge of \( I \), i.e., \( I \) is not in the common language of \( X \), or \( Y \). We first consider the interaction of arbitrary \( \hat{C} \) and honest party \( X \) and show a simulation that uses the strategy of \( \hat{C} \) to simulate \( X \) as follows:

**Simulation** \( M_{\hat{C}}(\langle G, G_x, G_y \rangle) \)

- Step 1: Let \( \hat{C} \) send the first message.
- Step 2: Send random \( R \) and \( K \).

Based on the proven verifier soundness, \( \hat{C} \) can not distinguish this simulation from an honest interaction with \( X \). Hence, no knowledge is leaked from \( X \) to \( \hat{C} \).

Similarly, the message of \( Y \) can be easily simulated by \( \hat{C} \) by sending random \( B_1 \) and based on verifier soundness, the simulation is computationally indistinguishable from the interaction with \( Y \).

\[ \blacksquare \]

### 6.4 Applications

**Group Keys and Federated Anonymity**  
Group keys enable entities in the same group to authenticate each other. There exist many approaches to group key management, but we focus on two well known models and demonstrate the benefit of using proofs on common languages for trust establishment based on the selected model.

We consider the basic approach of having a single group secret known to each entity within a group. In this case, different groups have different unique secrets associated with their entities. Nonetheless, if we consider our common languages paradigm, these secrets can be used to enable, not only intra group authentication, but inter group authentication as
well. Groups can maintain their own secrets but they can be federated if there exists some common values that are related to the secrets of these groups. The federated groups then can use proofs on common languages to establish trust among entities within and among themselves based on their common language. Consider entity $P_1$ and $P_2$ in groups $G_1$ and $G_2$ respectively. Also Consider entity $V$ in Group $G_1$, then:

- Both $P_1$ and $P_2$ can establish common language based trust with $V$. Note that $P_1$ and $V$ share the same group secret, while $P_2$ and $V$ have different secrets on a common value $I$.
- $V$ will not be able to distinguish $P_1$ from $V$ when both authenticates themselves to $V$.

The above two observations demonstrates the usefulness of proofs on common languages, especially that it provides anonymity across the federated groups. It then follows that a member of a group can not tell if an authenticated entity is inside its group or not, we call this “Federated Anonymity”.

An alternative to having a single group secret known to each entity is one where multiple keys are assigned to enable each entity to communicate either directly or indirectly (via intermediate entities, aka delegates) with all the other entities in the group. An example of this approach is grid based keys. The entities in the group are organized in a grid, and the grid location of each entity determines what keys an entity gets and the set of entities it can communicate directly with. For example [51], an entity shares a key with all other entities that are in the same row (likewise, in the same columns). That is, given $m$ entities, there
Figure 6.1: Grid-based Key Management.

are $O(\sqrt{m})$ keys.

Considering the use of proofs on common languages in this setting may lead to significantly decreasing the number of keys needed at a node. Using common languages and proofs of membership in common languages, one can achieve an implementation that requires as little as one key at each node to achieve connectivity across the grid. As shown in figure 6.1, $P$ need not know the secrets of $P_1$ and $P_2$, it only needs to have a common value corresponding to the secrets of $P_1$, $P_2$ as well as its own secret.

**Proofs on Common Languages for Delegability**  Consider three entities $A$, $B$ and $C$. $A$ may be interested in requesting and acquiring service from $C$, but it may not have necessary resources or direct reachability to $C$. Moreover, $A$ may not be interested in directly interacting with $C$ for privacy and anonymity considerations. On the other hand $B$ can easily reach out for $C$’s services.
Assume we introduced the common language paradigm in this setting, then \( A, B \) and \( C \) can have three independent languages based on three different secrets. Let input \( I \) be a common value in the languages of the three entities. Then \( A \) can use \( B \) as its delegate to contact \( C \) as follows:

- \( A \) interacts with \( B \) on \( I \) for proving membership in a common language, hence establishing common language based trust with \( B \).
- \( A \) then submits its request for service to \( B \).
- \( B \) establishes common language based trust with \( C \) on \( I \) using its own knowledge, and requests service from \( C \).

In the above scenario \( C \) can not detect that \( B \) is requesting service on behalf of \( A \). This is because \( C \) has no way of knowing that \( I \) is also in the language of \( A \). This can be particularly useful for applications such as federated cloud authentication.
Chapter 7: Secrets for Lifetime: An Application

‘Pervasive computing’ (or ‘ubiquitous computing’) is a growing trend in which computing and communication are seamlessly integrated with end users. Various technologies, devices, and networks facilitate seamless computing, communication, collaboration, and commerce-related functionality for end users. This is made possible by embedding sensors, controllers, devices and data into the physical world. Trust, Anonymity and Privacy preservation are known as major factors to the acceptance and success of pervasive computing systems. Device networks underlying these systems have to provide secure and efficient authentication that is privacy preserving, as well as being sensitive to the pervasive and distributed nature of these systems.

Authentication is used to establish trust among devices based on cryptographic secrets owned by these devices. Nonetheless, the way in which these secrets are used for bootstrapping trust has a direct consequence on the key management overhead of these systems. Conventional cryptographic protocols that make direct use of the devices’ original secrets constantly leak information about these secrets. This follows from the computational security assumptions underlying practical cryptography. The problem affects both symmetric and asymmetric secrets but is particularly crucial for symmetric secrets cryptography which is the ubiquitous choice of most device networks. Over time, periodical key updates are necessary to maintain the security of these systems, which incurs significant overheads,
especially for device networks—whose numbers, size, and day-to-day influence is growing rapidly. Solutions that automate and/or reduce key management are therefore important. Ideally, due to the pervasive nature of these systems, one would like to maintain the devices’ secrets for lifetime while being able to use them to establish, restore or maintain trust. This can be achieved if such secrets are used in a way that leaks zero knowledge regardless of how many times these secrets are used, which extends the secrets lifetime to the entity’s lifetime.

Two other desirable properties for Pervasive Computing systems are privacy and anonymity. Most of the current trust models are identity based, which implies that in order for one device entity to trust/authenticate another, it needs to know the other entity’s identity. Hence, there exists an inherent trade off between trust and privacy in the current security models.

In this chapter, we study conditions for lifetime secrets and propose an application for bootstrapping trust based on lifetime secrets in device networks. This application uses the common language model for traditional zero knowledge proofs of knowledge, and demonstrates the improved soundness and increased efficiency that the common language model introduces to traditional proof systems.

### 7.1 Lifetime Secrets for Device Networks

A secret is a *lifetime secret* for a cryptographic protocol if it can be used indefinitely by its owner(s) in the protocol without the secret being leaked. Most conventional protocols use secrets to carry out cryptographic operations (say for identification, authentication, repudiation, confidentiality, etc.) and reveal the output explicitly, and thus their secrets are leaked with continued use and are not lifetime. When secrets are not lifetime, ongoing secret update is necessary.
7.1.1 Conditions Necessary for Lifetime Secrets

**Assumptions:** We make the following assumptions about the problem for parts (a) (b) and (c):

- Consider a secret $s$ that is provided to entity $E$. (i.e., we will not consider the protocols or methods of generating $s$ and its effect on the secrecy of $s$)
- Assume a single cryptographic protocol $A$ through which secret $s$ is being used by $E$ to realize security property $Q$.
- There exist other entities, some of which may be faulty/byzantine who interact with $E$ through $A$.
- It is required for $E$ to continue using $s$ for lifetime through $A$.

Let “Time” be the shortest time needed to break the computational hardness assumptions underlying $A$, let “leak” denotes the maximum amount of leakage from $s$ through a single session of $A$ and “BoundLeak" the maximum amount of leakage that does not require an update on $s$. then a necessary condition would be:

$$( (\text{Lifetime} < \text{Time}) \land (\sum_{\text{time}} \text{leak} \leq \text{BoundLeak}) ) \lor \text{PerfectSecrecy} )$$
$$\land \text{PerfectBackwardSecrecy})$$

To provide Lifetime guarantees for secrets we need to model the security of $A$ assuming an adversary whose computational power is upper bounded by “lifetime”. In this sense, if lifetime denotes indefinitely, then the adversarial model used must assume computationally
unbounded capabilities for the adversary. I.e., protocol A must provide perfect secrecy (since the adversary has infinite time to try and break the protocols, including commitments or factoring problems...etc).

Alternatively, we could consider computationally bounded adversaries (the practical cryptographic model) as long as we bound “lifetime” to the time complexity of the hardness assumptions underlying A. Practically “lifetime” is bounded as it is usually related to an entity/device lifetime or network lifetime. The secrecy of Devices’ secrets are usually not crucial after the network and its data is no longer relevant or of value. Nonetheless, one can argue that the data may still be relevant long after the network is gone.

Hence, to give a lifetime guarantee on s, it is necessary to bound this guarantee to be strictly less than the “time” needed to break the computational hardness assumptions underlying the cryptographic primitives used in A. Recall that the strength of the secrecy assumptions of A are controlled by numerous factors, including but not limited to: secret-s/keys sizes, the random generators used and the randomization domain.

Next, we focus on how much is leaked from s through a single session of A. It is necessary to guarantee that over the defined “lifetime” period, the aggregation of these leaks does not mandate an update, i.e., the total leakage over “lifetime” is less than or equal to “BoundLeak”.

Another concern would be if s is used to construct new secrets through A, then a necessary condition is to ensure that if those secrets were compromised then nothing is learned about s (Perfect backward secrecy). In case the derivation of s was of concern, we should have included Perfect Forward Secrecy in the necessary conditions as well (This ensures that if s is derived from other keys it will not be compromised if those keys are compromised in the future).
Although the Zero Knowledge (ZK) property is an important enabler of lifetime keys, we believe it should not be a necessary condition. The reason is related to how long is “lifetime” and the pattern of use of these keys. In specific, keys can be trivially lifetime if used only once (ex: Vernam’s One Time Pad). Hence we preferred to have ZK protocols in the list of sufficient conditions.

7.1.2 Conditions Sufficient for Lifetime Secrets

- Perfect Secrecy.

- Perfect Zero Knowledge argument (Assuming the existence of perfectly hiding computationally binding commitments)

Perfect ZK requires that the amount of information leaked be zero, regardless of the computational power of the observer. In contrast, a computational zero-knowledge protocol may leak information, but if a Turing machine (Distinguisher) only has polynomial time to inspect the messages exchanged between the prover and the verifier, then the amount of information that it can gain from them is negligible.

- (Lifetime<Time) and Computational Zero Knowledge schemes secure against active adversaries. (Possible adversarial activities can be protocol/application specific).

Computational ZK ensures that the prover, even with infinite computational power, cannot convince the verifier to accept a false theorem. However, the proof itself is only computationally secure: i.e., if the verifier (or anyone overhearing the execution of the protocol) ever breaks the cryptographic assumption, say after 100 years, additional knowledge about the proof can be extracted, hence the condition (Lifetime<Time).
7.1.3 Lifetime Secrets for Device Networks

Device networks can benefit from computational ZK, especially that the lifetime of such networks are strictly bounded due to its resource constrained nature. Providing lifetime guarantees in such networks would be greatly simplified by using the asymmetry inherent in Device networks (base station-nodes), which has the potential to simplify the ZK model and potential attacks. One can assume that lifetime secrets are preshared and that it can be used to create new session secrets. In this case protocols of authenticated key exchange that make use of lifetime secrets should be zero knowledge and should guarantee perfect backward secrecy.

When privacy and anonymity are not required, the standard ZK protocols can be used for establishing identity-based trust. Nonetheless, the cost of such protocols, including the multiple rounds requirements for security, has been a challenge of realistic implementations for resource constrained device networks. We propose a ZK-based solution that establishes identity-based trust via a common language defined in terms of the identity as well as an additional secret which enjoys efficiency in terms of reduced number of rounds (to as little as one round) as well as the additional benefit of "key transport". We demonstrate an application of this solution for bootstrapping device networks in the field. In this solution zero knowledge (ZK) is not only used for identification (based on the identity and the additional secret), but to communicate group permission secrets. To the best of our knowledge, ZK schemes have not been used in this way before. We show that it suffices for each device to share pre-deployment only two keys with a well known device, namely the base station. These two keys establish a common language between the device and the base station and are persistent; i.e., they do not need to be updated since they will be always used in a zero knowledge way. The asymmetry inherent in device networks (BS-nodes) makes for secure
choice of the ZK randoms as well as avoiding critical concurrent or resettable attacks which makes the use of lightweight ZK practical for Device Networks.

Conversely, when privacy and anonymity is of concern, our approach for authentication is not based on identities, but rather on common languages. We propose to bootstrap trust among entities based on common languages using zero knowledge interactive proofs on common languages. This guarantees that nothing is leaked about the entity’s secrets. Trust based on common languages can be established among entities using symmetric or asymmetric secrets as long as these secrets correspond to common values on which both entities has computational ability or knowledge. This authentication approach is not based on identities and is zero knowledge, we also demonstrate that it can be realized by efficient protocols, using as few as two messages. which makes it particularly useful for Device Networks and Pervasive Computing Systems. The established trust can then be used for a host of applications such as group key management, deniable authenticated key exchange, secret handshakes and more. All in terms of common languages as a basis of trust.

7.1.4 Models of Secrets Deployment in Device Networks

We consider two possible system models for device networks that make use of lifetime keys. The assumptions we made for the two models restrict the use of lifetime secrets to only authentication and authenticated key exchange (AKE). It then follows that for a protocol to preserve secrets for lifetime it has to leak zero knowledge, i.e., it has to be proven secure against the adversarial models of ZK protocols as well as AKE adversaries that challenge forward and backward secrecy.
The Asymmetric Deployment Model:

This model satisfies the following:

- Each device shares one or more unique keys with a well known single device, namely the base station.
- These keys should be persistent for lifetime to bootstrap the devices with new group secrets or make updates or deletions of secrets at the device.
- These keys are only used in communications with the base station.
- These keys can be used to create session keys with the base station.
- There is only one base stations and it cannot be reset, tampered with or compromised.
- The base station can make optimal choices of its randoms and the secrets. Theoretically it can be modeled as an all powerful party with unbounded computational power. Moreover, it can control the number of concurrent sessions and may even limit it to one session at a time.
- Lifetime keys are only used for authentication or authenticated key exchange (AKE), where the devices act as the authenticator to the base station.

The Symmetric Deployment Model:

This model satisfies the following:

- Each device shares one or more keys with one or more devices.
- Keys can be shared across multiple entities.
• These keys should be persistent for lifetime and are only used for authentication or authenticated key exchange (AKE).

• Devices should be able to use these keys to create session keys as needed.

• Some devices may become faulty/byzantine.

• Devices are computationally bounded.

7.1.5 Adversary Model for Lifetime Secrets in Device Networks

When considering lifetime keys for device networks, it is necessary to accurately specify the adversary model that should be considered in the security analysis of any protocols used for device networks for which keys are assumed to persist for lifetime.

The asymmetric system model does not require a demanding adversarial model. A secure ZK protocol based on the standard definition (i.e., sequential ZK, not necessarily concurrent ZK or resettable ZK) (even weaker than Dolev-Yao) would suffice. The reason is that the base station is a powerful control unit that can selectively choose the sessions it actively participate in, carefully selects its randoms and composes its messages as well as forcing sequential execution of these sessions if needed. In this model, the base station acts as a single prover and the devices act as the verifiers. Hence, no two verifiers can engage in concurrent sessions with the base station and no adversary can reset or compromise the base station (as per the assumptions). Moreover, the (BS-Device) preshared secrets are unique.

Nonetheless, we need to guarantee that keys constructed based on lifetime secrets do not leak information about these secrets and vice versa, directly or indirectly. We add to the Dolev-Yao adversary the ability to compromise session keys of other (non-matching)
sessions, thereby modeling known-key attacks. This is important to guarantee that session keys cannot be related across different sessions, which may leak information about the lifetime secrets used to establish them. We further strengthen the adversary by giving it the ability to compromise long-term keys after the session, as well as allowing it to reveal the local state (e.g., the random numbers) generated in other sessions. This is important to determine the effect of long-term key compromise on other sessions. Such properties are captured in the CK security model which we suggest for the Asymmetric deployment model.

For the Symmetric deployment model there are challenging considerations to realize lifetime keys. One should consider concurrent ZK as well as resettable ZK and because the devices are modeled as computationally bounded entities and are prone to different forms of attacks and compromises, especially when lifetime keys are used to establish session keys. We choose the universally composable multi-party computation security model (UC-model) for ensuring that the intuitive notion of security (ZK) is captured with respect to any practical adversarial behavior under consideration.

The UC-model in [23] considers the protocol in isolation, but guarantees secure composition. Within this framework and the underlying adversarial model, protocols are guaranteed to maintain their security in any context, even in the presence of an unbounded number of arbitrary protocol instances that run concurrently in an adversarial controlled manner. This is a useful guarantee, that allows arguing about the security of cryptographic protocols in complex and unpredictable environments such as Device networks.

The model assumes a set of interacting computing elements, representing the parties running the protocol. Formally, these elements are modeled as interactive Turing machines (ITMs). An additional ITM represents the adversary, who controls a subset of the parties,
and has control over the scheduling of message delivery, subject to the synchrony guarantee. Another algorithmic entity, called the environment machine is added to represent whatever is external to the current protocol execution (this includes other protocol executions and their adversaries, human users, etc.). The parties and adversary interact on a given set of inputs and each party eventually generates local output. The concatenation of the local outputs of the adversary, and all parties is called the global output. The environment interacts with the protocol execution twice: First, it hands arbitrary inputs of its choosing to the parties and to the adversary. Next, it collects the outputs from the parties and the adversary. Finally, the environment outputs a single bit, which is interpreted as saying whether the environment thinks that it has interacted with the protocol or with the ideal process for \( f \). Now, we say that protocol \( A \) securely evaluates a function \( f \) if for any adversary \( M \) there exists an “ideal adversary” \( L \) such that no environment can tell with non-negligible probability whether it is interacting with \( M \) and \( A \) or with \( L \) and the ideal process for \( f \).

7.2 Bootstrapping Trust in Device Networks via Lifetime Keys

When privacy and anonymity are not required, the standard ZK protocols can be used for establishing identity-based trust. In this section we demonstrate that we can make use of common languages for proofs of identity to produce more efficient protocols and explore new uses of the standard proofs. Focusing on symmetric secrets, this model assumes that both the prover and the verifier share some secrets (their common language) and explores the use of these secrets to allow the honest verifier to gain an advantage in terms of being able to verify the prover’s proof with high confidence which implies less rounds, as well as enabling the prover to communicate new secrets to the verifier.
We demonstrate an application of this model for bootstrapping device networks in the field. In device networks, trust must often be established in the field despite limited a priori knowledge of the network and the possibility of adversaries in the network environment. We present a solution to the problem of bootstrapping trust that uses proofs of identity on common languages. Our solution is minimal in the sense that it circumvents ongoing maintenance of security material. Specifically, security material is communicated to members of a device group just once by using zero knowledge identification over a common language established by the preshared secrets. Devices in the group may henceforth securely verify each other as well as initialize mutual keys for confidentiality without needing to update that security material over time. In its basic form, the solution uses a base station to communicate the security material for group membership verification. The solution allows for scaling by letting the base station hierarchically delegate the task of bootstrapping to subordinate trusted nodes.

7.2.1 Motivation

System security today relies essentially on the use of shared keys. Key management however incurs significant human involvement. For the case of device networks—whose numbers, size, and day-to-day influence is growing rapidly—it is impractical to sustain the status quo. Solutions that automate and/or reduce key management are therefore important.

An extreme position is to explore solutions that use no shared keys. Along these lines, recent work, including some of ours [67, 4], has considered the alternative of eschewing shared keys via physical layer security. The central idea is to realize a seminal result from information theory [74], which has been underexploited in practice to date, via a number of
physical primitives for realizing security properties such as confidentiality, authentication, etc. without relying on shared keys.

Nonetheless, consideration of bootstrapping these physical primitives for security reveals an interesting chicken-and-egg problem: Even though physical primitives themselves do not use shared keys, bootstrapping—as well as maintaining and restoring—security material for enabling physical primitives itself assumes that inter-node trust relationships have been established beforehand. For example, instantiation of confidentiality, say using dialog codes [4], assumes a mechanism by which the sender and the receiver can authenticate each other and agree to launch cooperative jamming. Shared keys may be used for bootstrapping confidential communications. Alternatively, we could use physical signatures/fingerprints as the security material for authentication, if we wished to avoid shared keys. But in that case, the instantiation of physical signatures in turn assumes there is a mechanism for coordination between nodes that trust each other whereby the signature is learned. In sum, bootstrapping of shared-key-free physical primitives itself assumes shared keys!

For what sorts of trust relationships, then, should we bootstrap a device with shared keys? One is a device’s trust in the network that it itself is to be a part of. And two, its trust in one or more devices in the network with which it is to communicate as part of some common application. While the former can be bootstrapped before devices are deployed, the latter is typically bootstrapped after devices are deployed, as which particular devices will end up communicating with each other is often not known a priori. In any case, both of these sorts can be abstracted as trust in some group of devices. And the shared keys established for the former can be used to establish shared keys for the latter.
In this chapter, we present a solution to the problem of trust bootstrapping in device networks. As our goal includes reducing the key management overhead, we adopt the position that all of the shared key material available a priori for trust establishment is in some sense minimal. In particular, for the first sort of trust, nothing is needed a priori, since this can be performed pre-deployment. For the latter sort of trust, we show that it suffices for each device to share pre-deployment only two keys with a well known device, namely the base station. These two keys are persistent; i.e., they do not need to be updated.

The security of using shared keys typically degrades over time. The idea central to our eschewing the need to update these two pre-shared keys is to use them in a zero knowledge fashion. More specifically, we use them with a zero knowledge identification scheme, to serve as the basis for communicating security material and thereby to enable the sharing of secrets. Our solution is distinguished by the following properties:

- In its basic form, it uses a base station to communicate to group members the relevant security material (i.e., group permissions). A straightforward extension of it allows for scaling by letting the base station hierarchically delegate the task of bootstrapping to subordinate trusted nodes.

- It is collusion resistant in the sense that compromising any device $j$ does not compromise the network’s ability to bootstrap devices other than $j$ with new group permission security material.

- It uses standard cryptographic constructs, so it is flexible enough to allow plug in of different realizations of these constructs, including ones which are more efficient or are more secure.
• It uses these constructs in a non standard, efficient way. That is, it uses zero knowledge (ZK) not only for identification, but to communicate group permission secrets, and to construct Diffie-Hellman keys (DH) without exchanging key parts. To the best of our knowledge, ZK schemes have not been used in this way before. Efficiency follows from eschewing the need for many rounds in the ZK scheme as well as reducing the overhead of DH.

• It bootstraps trust in levels, by developing a trust hierarchy wherein trust can propagate top down.

7.2.2 Related Work

There is rich literature on ZK identification schemes. ZK research in the context of resource constrained networks has focused on optimizing the implementation of these schemes. An illustrative example [2] is a modified version of the Guillou-Quisquater (GQ) identification scheme that is used in conjunction with the ÎŒTESLA protocol [62] to authenticate the base station using a group of nodes. Another example [50] refines the Feige-Fiat-Shamir ZK scheme to reduce the number of challenge-response rounds so as to speed up the authentication process; the refinement uses bursts of parallelism while maintaining serial execution of Feige-Fiat-Shamir in order to preserve the ZK property. Efficient hardware implementation of the ZK identification schemes have been presented in the context of authenticating RFID-tags [72, 9]. To the best of our knowledge, however, none of the existing works in the ZK literature on device networks have considered the use of ZK for communicating security material, as we have for the purpose of communicating group permissions as part of the bootstrapping process.
A variety of efforts have attempted to examine the use of public key cryptography in resource constrained networks [65, 58, 57], and many have focused on the special case of ECC [70, 55]. There are a few effort related to optimizing DH implementations for device networks [52, 55]. To the best of our knowledge, again, we are not aware of previous work that makes use of DH keys in conjunction with the ZK identification scheme such that key agreement can be accomplished without explicit exchange of key parts. The approach of assuming pre-shared key parts between individual nodes and BS has been used previously [27], to update the mutual keys by regularly broadcasting fresh key parts, but this lacks authentication. Moreover, unlike our protocol, the updates are sent in the clear, which is vulnerable to timing attacks, especially if the same key part is to be used by all nodes in the network.

7.2.3 The Trust Bootstrap Problem

Network Model  Given is a set of device nodes and a base station (BS). Nodes can communicate with one another and with BS; communication with BS may involve one or more hops. The network is a programmable “fabric” in the sense that multiple applications (aka, groups) may coexist. Associated with each application is a subset of network nodes, which are to execute that application. We assume that application groups are not known a priori, i.e., before the network is deployed.

Attacker Model  Our attack model includes the Dolev-Yao attacker: the attacker can overhear, intercept, and synthesize any message but is limited by the constraints of the cryptographic methods used. In addition, our attack model includes the compromise of any node. When a node is compromised, its state and all of its secrets and programs are
available to the attacker. The attacker may use the material and programs of its own, for instance, to collude with other compromised nodes and/or impersonate other nodes. During a protocol execution, the attacker may execute multiple instances of the protocol scheduled in an arbitrary way. We assume however that BS cannot be compromised.

**Trust Bootstrap Problem** Given is a subset of nodes, \( G \), in the network \( W \). Required is a protocol to initialize security materials (e.g., shared keys) in each node in \( G \) so that *only nodes in \( G \) can successfully test each other’s membership in \( G \).* That is, a node \( j \) can prove that \( j \) is in \( G \) to a node \( i \) only if \( j \) is in \( G \) and, conversely, \( i \) can verify that \( j \) is in \( G \) only if \( i \) is a member of \( G \), thus, trusted nodes in \( G \) can authenticate communications from other trusted nodes in \( G \).

**Solution considerations** Standard approaches to establish a secure group include (i) using pre-shared keys or keying materials, (ii) group nodes exchanging information with their immediate neighbors, or (iii) group nodes exchanging information with computationally robust nodes, for example, BS.

The number of pre-shared keys per device typically depends on the size of the network, so scalability can be a problem with pre-shared keys. The idea of random key distribution yields indeterministic solutions in terms of sufficing for groups, which makes it unsuitable for our case since node groups are not known a priori. The approach of exchanging information with immediate neighbors is inappropriate given the system model, since nodes can not trust any other node in the fabric a priori. The approach of using BS is feasible, in part because the BS is always trusted. The use of BS however risks becoming a bottleneck and a single point of failure. Ideally, one would like to design alternatives where BS can delegate its task to nodes it trusts so as to reduce the overhead of potentially multi-hop communications as well as to obtain robustness to BS failure.
7.3 Zero Knowledge Approach for Bootstrapping Trust

Our bootstrapping approach makes use of pre-shared keys in the following minimal sense. Every node shares a symmetric key with BS just once. BS uses this key to deliver group permission secrets to the node. The node in turn can use these permissions to establish trust with other nodes in the groups. If a pre-shared key is used to directly encrypt information, its security degrades over time and so it should not be used indefinitely. For the security of the key to be preserved forever, our approach builds upon that of Zero Knowledge (ZK).

<table>
<thead>
<tr>
<th>$i, j$</th>
<th>Nodes in $W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$BS$</td>
<td>Base station</td>
</tr>
<tr>
<td>$r, n$</td>
<td>Randoms</td>
</tr>
<tr>
<td>$S, v$</td>
<td>Main secrets, unique and private to each node</td>
</tr>
<tr>
<td>$g$</td>
<td>Group permission secret, local and private to members of $G$</td>
</tr>
<tr>
<td>$k(m)$</td>
<td>Encryption of $m$ using a symmetric key $k$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Group generator</td>
</tr>
<tr>
<td>$x$</td>
<td>Large prime that is global and public</td>
</tr>
<tr>
<td>$t$</td>
<td>Time stamp</td>
</tr>
<tr>
<td>$f, T$</td>
<td>Special purpose functions</td>
</tr>
<tr>
<td>$k_a$</td>
<td>Diffie-Hellman key part, constructed using the secret exponent $a$</td>
</tr>
<tr>
<td>$k_{a,b}$</td>
<td>Diffie-Hellman key, constructed using the secret exponents $a$ and $b$</td>
</tr>
</tbody>
</table>

Table 7.1: Bootstrap Protocol Notation

Recall the standard form of the ZK identification schema [44, 10], cf. Figure 7.1: In $ZK[r, n, S]$, a prover $P$ sends a commit, $f(r)$ where $f$ is a one way function, and then based on receiving the challenge $n$ from the verifier $V$, sends a response $f(r, n, S)$ to $V$ in order to
prove to V that P knows S. This process does not reveal any knowledge including S beyond the validity of the proof to any node other than P.

Using ZK lets us guarantee that the symmetric key shared between BS and each node does not degrade over time. More specifically, our solution extends the standard ZK identification schema to also securely communicate group permission secrets to nodes in G. Moreover, our solution is designed such that BS is not aware of group/pairwise keys established within the groups, but rather just the permission secrets that enable nodes to authenticate their membership in the group.

7.3.1 Trust Levels: The Architecture

Our solution bootstraps trust in levels, resulting in a trust hierarchy. At the top of the hierarchy is BS, which represents the device network. Lower levels of the hierarchy correspond to trust in members of groups and/or subgroups in the superset of groups G in W. Corresponding to the levels in the hierarchy are different types of secrets, namely: main, permission, group, and pairwise, defined as follows.
Main secrets  Each node $j$ has two unique secrets, $v$ and $S$, that $j$ shares with the BS. These are persistent secrets that can be used to bootstrap $j$ any number of times for membership in a number of groups of $W$.

Permission secrets  Each group $G$, $G \in G$, has a single unique secret $g$ shared by all nodes $\in G$, which is used solely for authenticating membership in $G$. The group permission secret $g$ is not used directly for encrypting any exchanges, hence it can be used infinitely in $G$ as long as nodes of $G$ are not compromised.

Group secrets  Each group $G$, $G \in G$, has zero or more secrets, $\hat{S}$, that are shared by all or some nodes $j \in G$, and used say to authorize service requests, provide services, relay communications, receive results, or instantiate physical primitives. Nodes may use these secrets explicitly in encryption, hence they may need to update them from time to time.

Pairwise secrets  Each node pair, $i, j \in G$, has zero or more pairwise keys. Again, nodes may use these secrets explicitly in encryption, hence they may need to update them from time to time.

Figure 7.2: Levels of Trust
**First Level of Trust (TL1)** For each node $j$, BS can mutually authenticate.

- TL1 is established pre-deployment by configuring $j$ with the main secrets $v$ and $S$ shared by the BS.

- $S$ is used in a ZK way only, s.t. while $j$ is capable of extracting knowledge from the ZK proof using $S$, for all nodes other than $j$, the proof is zero knowledge. Also $v$ is never used directly in any exchanges.

- BS can use the main secrets to deliver permission secrets to $j$ for joining a group $G \in G$.

**Second Level of Trust (TL2)** For each node $j$, nodes in $G \in G$ can authenticate $j$ iff $j$ is in $G$, based on TL1:

- TL2 is established in the field, based on knowing the permission secret $g$.

- $g$ can be updated if some node of $G$ is (suspected of being) compromised.

- BS maintains the permission secrets of each defined group $G \in G$.

**Third Level of Trust (TL3)** For each node pair $i, j \in G$, $i$ and $j$ can establish group secrets and/or pairwise secrets, based on TL2.

- TL3 is established in the field.

- Group secrets, $\hat{S}$, can be communicated as part of the authentication of $i$ by $j$, or alternatively using pairwise keys.

- Pairwise keys can be used to update $g$ as needed.
7.3.2 Setup and Secrets Generation

- ZK and DH parameters: Depending on the particular ZK and DH protocol realizations chosen (i.e., ECC vs standard), relevant parameters are generated. The parameters are global and generated pre-deployment.

- $S$: A node’s symmetric secret, pre-configured and shared with BS. The security properties of $S$ are dependent on the particular ZK scheme realization that is selected.

- $v$: A symmetric randomly chosen secret, $1 \leq v \leq x - 1$, that is pre-configured in the node and shared with BS.

- $g$: Let $x$ be the DH large prime. The permission secret $g$, $1 \leq g \leq x - 1$, is generated by BS for any new group $G$.

- $k_{r_j}$: For each node $j$, a static DH key part is generated pre-deployment using some random secret exponent, $r_j$. This is used only in TL3 to reduce the overhead of DH.

7.3.3 Trust Protocols

TL1 is established pre-deployment, hence every node $j$ in $W$ is capable of verifying BS exchanges. Nonetheless, any node $i, i \neq j$, may be able to verify a BS exchange to $j$ but would not be able to extract any knowledge from it.

A protocol using TL1 to establish TL2  The following protocol uses the secrets $S$ and $v$ established in TL1 to establish TL2 by communicating the permission secret $g$ of a group $G \in G$ to node $j$. 

171
\[ r' = T(r) \]

**TL2 Protocol**

\[
BS \rightarrow j : ZK[r,n,S] ; k_{r',v}(g,j,n,t)
\]

Let \( T \) be some transformation mapping a random number from one range to another while preserving the randomness property. To bootstrap a node \( j \), the protocol uses a standard ZK identification session, \( ZK[r,n,S] \) between \( BS \) and \( j \). The ZK commit in terms of the random \( r \) along with the challenge \( n \) and main secret \( S \) formulates the \( BS \) response.

As shown in Figure 7.3, \( BS \) sends along with the ZK response, an encrypted message, \( k_{r',v}(g,j,n,t) \), to \( j \) containing the permission secret \( g \) along with some other parameters to tie the message to the proof. The encryption key \( k_{r',v} \) is a DH key, formulated using \( r' \) and \( v \) as the secret exponents. Note that none of the DH key parts are exchanged in public. \( r' \), the
value of the ZK random $r$ mapped onto the DH ring using $T$, along with the precomputed DH key part using $v$, will never be known except to $j$. $j$ uses its main secret $S$ to extract $r$ during the proof verification process. Note that $S$ is known only to $j$ and BS, thus for all nodes other than $j$, the proof is zero knowledge. $j$ then uses $r'$, along with its local precomputed key part in terms of $v$, to construct the key $k_{r',v}$ for decrypting $k_{r',v}(g,j,n,t)$.

**Property 7.1** The value of $r'$ is random in the DH range: $[1,x-1]$.

**Theorem 7.1** The zero knowledge property of ZK[r,n,S] is preserved by Protocol TL2.

**Proof outline** Assume the existence of an efficient oracle $O$ that, given an encrypted message, returns the key used for encryption. An adversary $\hat{A}$ could query $O$ for $k_{r',v}(g,j,n,t)$ and be given $k_{r',v}$. Note that the key’s exponent, $r'.v$, would be some random in the DH ring, since both $r'$ and $v$ are randoms. Finding $r'.v$ implies solving the Discrete Log problem, which is assumed to be intractable. We hence have a contradiction that proves the hypothesis.

**Theorem 7.2** It is computationally hard to deduce the secret $v$ of any node $j$ regardless of how many times Protocol TL2 is executed.

The proof of Theorem 7.2 closely follows that of Theorem 7.1.

**A protocol using TL2 to establish TL3** TL3 lets a node in $G$ prove its membership to any other member node. This allows members of $G$ to exchange group and pairwise secrets as need be.

**TL3 Protocol**

```
 i → j : k_{r_i}, k_{r_i,s}(i,j,t,\hat{S})
```
Node $i$ sends a message encrypted with a DH key $k_{r_i,g}$ constructed using $g$ and its private random $r_i$ corresponding to its precomputed static DH key part $k_{r_i}$. Note that the DH key part in terms of $g$ is never communicated in the clear, hence, only $G$’s nodes are capable of composing the key $k_{r_i,g}$. This simple one-way exchange serves for two-way membership authentication due to the ability of both $i$ and $j$ to successfully construct such a key. Note that we do not require that nodes store the DH key parts of other nodes, so, $i$ has to send its precomputed key part along with the membership proof. $\hat{S}$, a group secret communicated from $i$ to $j$ can be used in a number of ways. Note also that based on successful membership authentication, $i$ and $j$ can establish a pairwise key using the precomputed DH key parts.

### 7.4 Implementation and Analysis

In this section, we propose possible realizations of Protocol TL2 using different ZK identification schemes that cater to devices which are resource constrained or which demand a high level of security. We then develop a simple cost model for estimating the overheads of Protocols TL2 and TL3. Finally, we discuss the distinctive properties of our solution, towards arguing that the potential merits of our approach justify the cost overheads, which per se are not high.

#### 7.4.1 Protocols Realizations

We consider three realizations for Protocol TL2. One uses the Guillou-Quisquater (GQ) ZK identification scheme [46], which is based on the hardness of the RSA problem, the other two use the Elliptic Curve (ECC) versions of Schnorr (SC) [72, 68] and Okamoto’s (OK) ZK [9, 59] identification schemes, which are based on the discrete log assumption. The base versions of GQ-ZK and SC-ZK are known for their efficiency, while OK-ZK
scheme is known for its security features. Device and deployment constraints would dictate which of these to use in particular contexts.

**GQ TL2 Protocol**

GQ-ZK requires some parameters that are global to the network: \( w \), the RSA modulus, and \( e \), a prime RSA exponent. Local parameters include the node’s main secret \( S \in \mathbb{Z}_w^* \) that is shared with \( BS \). \( \alpha \) is the chosen public DH generator. Let \( T \) be some transformation scaling randoms in the range \([1, w - 1]\) to the range \([1, x - 1]\) while preserving the randomness property. Note that numerous efficient implementations exist for similar transformations, also based on how both ranges compare, the transformation can be as simple as taking the modulus, i.e., mod \( x \).

**Protocol for TL2 establishment: GQ version**

\[
\begin{align*}
BS & \rightarrow j : r^e \pmod{w} \\
j & \rightarrow BS : n \\
BS & \rightarrow j : rS^n \pmod{w}, k_{r',v}(g, j, n, t)
\end{align*}
\]

In our protocol, \( BS \) acts as a prover while a node \( j \), which is to be bootstrapped to join a group \( G \), is the verifier. In the protocol, detailed below, \( BS \) selects a random \( r \in \mathbb{Z}_w^* \) and sends a commitment to \( j \) which replies back with the challenge \( n \). Finally \( BS \) sends its response to the challenge along with an encrypted message. The key used for encryption can be viewed as another challenge because it is not known to the node (i.e., not pre-shared), nor is it based on any key parts communicated in the clear. This key is constructed in terms of \( r' \) and \( v \), where \( r' \) is the mapping of \( r \) onto the DH group using \( T \). Hence, the only way \( j \) could construct this key is to use its secret \( S \) to extract the random from the \( BS \) proof.
Let $R$ be the commit, $C$ be the response, and $D \equiv S^e (modw)$. The GQ-ZK acceptance condition then is $C^e \equiv RD^p (modw)$. In our protocol, acceptance is reduced to checking if $R \equiv r^e (modw)$, which is attributed to $j$’s ability to learn $r$ from the proof.

**ECC-SC TL2 Protocol**

ECC-SC global parameters include: $q$, which specifies the finite field, $a,b$, which define an elliptic curve, $P$, a point on the curve of order $x$, and the cofactor $h$. Let $D = -S.P$, where $S$ is a main secret of $j$. The SC-ZK standard verification would be: if $C.P + n.D = R$ accept else reject, where $C$ is the response, $R$ is the commit and $n$ is the challenge.

**Protocol for TL2 establishment: ECC-SC version**

<table>
<thead>
<tr>
<th>BS $\rightarrow$ j</th>
<th>r.P</th>
</tr>
</thead>
<tbody>
<tr>
<td>j $\rightarrow$ BS</td>
<td>n</td>
</tr>
<tr>
<td>BS $\rightarrow$ j</td>
<td>$Sn + r (mod x)$, $k_{r,v}(g,j,n,t)$</td>
</tr>
</tbody>
</table>

On the other hand, our protocol uses $S$ to extract $r$ and verify that $R = r.P$. $j$ then uses its secret $v.P_1$ and performs a single multiplication by $r$ to derive the challenge key $k_{r,v}$. Notice that in this protocol version, ECC-DH is used, where $P_1$ is a public high order base point of the aforementioned chosen curve. If we choose the order of $P_1$ to be the same as $P$, then Property 3.2 would be a necessary condition for correctness, since we require anonymity of the key parts composing $k_{r,v}$. Note that in this case, no transformations would be needed for $r$. A safer option though would be to choose a different order for $P_1$, then a scaling transformation, similar to the one used in the GQ TL2 Protocol, would be needed to map $r$ from the range defined by the order of $P$ to that of $P_1$.

**Property 7.2** For DH generators $P_1$ and $P$ of order $x$, given $r.P$ for some index $r$, it is hard...
to find \( r.P_1 \) without knowledge of \( r \).

\[ k_{r,v} = r.v.P_1 \quad (7.2) \]

**ECC-OK TL2 Protocol**

For OK-ZK, the public global parameters are \( q \), which specifies the finite field, \( a, b \), which define an elliptic curve, \( P_1 \) and \( P_2 \), points on the curve of order \( x \), and the cofactor \( h \). In this case, our protocol uses a pair \((S_1, S_2)\) as a main secret.

**Protocol for TL2 establishment: ECC-OK version**

<table>
<thead>
<tr>
<th>BS ( \rightarrow ) j</th>
<th>( r_1.P_1 + r_2.P_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>j ( \rightarrow ) BS</td>
<td>( n )</td>
</tr>
<tr>
<td>BS ( \rightarrow ) j</td>
<td>( S_1n + r_1 \pmod{x}, \ S_2n + r_2 \pmod{x}, \ k_{r',v}(g, j, n, t) )</td>
</tr>
</tbody>
</table>

Let \( D = -S_1.P_1 - S_2.P_2 \), a node \( j \) accepts the proof iff \( C_1.P_1 + C_2.P_2 + n.D = R \), where \( C_1 \) and \( C_2 \) are the ZK responses of BS, \( R \) is the commit and \( n \) is the challenge. Here, we have two randoms \( r_1, r_2 \), so one choice would be to use a function \( f \) that maps the two randoms into a single random. This single random is then used by the node to construct the DH key, \( k_{r',v} \), for decrypting the message \( k_{r',v}(g, j, n, t) \). Note that it is safe to choose order \( x \) for the DH base point \( P_3 \), since even if a trivial isomorphic mapping from \( P_1 \) and \( P_2 \) onto the group generated by \( P_3 \) exists, then based on the DH assumption, given \( r_1.P_3 \) and \( r_2.P_3 \) it would be hard to compute \( r'.P_3 \), assuming \( f \) is not a trivial addition for example. Note that it is also hard to extract \( r_1.P_1 \) and \( r_2.P_2 \) from the commit in the first place.
\[ r' = f(r_1, r_2) \]  \hspace{1cm} (7.3)

\[ k'_{r,v} = r' . v . P_3 \]  \hspace{1cm} (7.4)

### 7.4.2 Security Features of the Realizations

OK-ZK is provably secure against impersonation under active and passive attacks. GQ-ZK and SC-ZK schemes are known for their efficiency, and also to be secure against impersonation under passive attacks, assuming honest verifiers. In addition, in [12], the authors extend the results of GQ-ZK to include security against active attacks based on the assumed security of RSA under one more inversion. The authors also provide such a proof for SC-ZK based on a corresponding discrete-log related assumption, they further extend their proof to establish security against impersonation under concurrent attack for both schemes. In our protocols, we only change the order in which the node verifies the proof, such that using \( S \), the node can extract the random and verify it against the commitment. Next, the node uses the extracted random to construct the challenge key for decrypting the group permission message. Note that no key parts are sent in the clear, and the challenge key is constructed using the ZK random and the main secret \( v \).

### 7.4.3 Performance Evaluation

Our trust hierarchy uses ZK identification, DH key agreement protocol and symmetric encryption/decryption. In evaluating the performance of protocols TL2 and TL3, if one were to consider a baseline for comparison which used only symmetric keys explicitly, one
would have to model the cost of operations which are performed on an ongoing basis to
update these keys. Rather than do so, we focus on estimating the overheads of ZK and DH
as they are used by the protocols. We show how the choice of the cryptographic constructs
and the pre-calculation of some values can decrease the expected overhead.

We define a cost function $\Psi$ for Protocol TL2. The cost of establishing TL2 for a node $j$ in
group $G$ is given by the following equation.

$$
\Psi_j(G) = \delta_{ZK} + \delta_{DH} + \delta_{sym} 
$$

(7.5)

Where $\delta_{ZK}, \delta_{DH}, \delta_{sym}$ define the costs of ZK identification scheme, DH key construc-
tion, and symmetric encryption/decryption, respectively. The cost of establishing TL2 for
a group $G$ of $n$ nodes is given by:

$$
\Psi(G) = \prod_j^n \Psi_j(G)
$$

(7.6)

Next, we define a cost function $\omega$ for Protocol TL3. The cost of establishing TL3 for a
node $j$ relative to a group $G$ of $n$ nodes is given by:

$$
\omega_j(G) = \delta_{DH} + \delta_{sym}
$$

(7.7)

Assuming ECC-ZK and ECC-DH, we can express $\delta_{ZK}$ and $\delta_{DH}$ in terms of the num-
ber of scalar point multiplications. Let $\Gamma$ be the cost of performing a single scalar point
multiplication. Using static pre-computed DH key parts brings down $\delta_{DH}$ to a single ECC
point multiplication ($\delta_{DH} = \Gamma$). In ZK schemes, we are interested in the verifier load share
only, which is the node to be bootstrapped. The verification of SC-ZK requires two point
multiplications ($\delta_{ZK} = 2\Gamma$), while OK-ZK requires four ($\delta_{ZK} = 4\Gamma$). This totals to an over-
head of three point multiplications for ECC-SC TL2 ($\Psi_j(G) = 3\Gamma + \delta_{sym}$), versus five for
ECC-OK TL2 ($\Psi_j(G) = 5\Gamma + \delta_{sym}$). TL3 has an overhead of a single point multiplication
\((\omega_j(G) = \Gamma + \delta_{\text{sym}})\). Note that for TL2, a node can have some pre-computed \(S, n\) values, where \(S\) is one of the node’s main secrets, and \(n\) is the ZK challenge, hence bringing down the overhead to two point multiplications for ECC-SC TL2 \((\Psi_j(G) = 2\Gamma + \delta_{\text{sym}})\), versus three for ECC-OK TL2 \((\Psi_j(G) = 3\Gamma + \delta_{\text{sym}})\).

Several optimized implementations exist for point multiplications [70]. For example, an elliptic curve implementation over a 192-bit prime field for MICAz motes [52] yielded a full scalar multiplication in 0.71 sec \((5.20 \cdot 10^6 \text{ cycles})\) when the base point is fixed and known a priori. Note that we are not restricting the protocols to specific ZK schemes, instead, the realizations serve to demonstrate the flexibility of adapting different constructs to the basic schema.

### 7.4.4 Features and Merits

Lastly, we discuss some of the distinguishing features of our bootstrap approach and the related protocols.

**ZK made more efficient** ZK protocols have lighter computational requirements than public key protocols since ZK uses an iterative process involving lighter transactions, thereby achieving its result with one to two orders of magnitude less computing power. A typical implementation might require 20–30 modular multiplications that can be optimized to 10–20 with pre-calculation. This is much faster than RSA [3]. In our protocols, more specifically in Protocol TL2, we use ZK in a way that makes a single round sufficient.

**Theorem 7.3** One ZK round is sufficient for TL2 establishment.

**Proof** For example, assume GQ-ZK. Let \(R\) be the commit defined in terms of the random \(r\), \(C\) be the response, and \(D \equiv S^r (mod \ w)\). The GQ-ZK acceptance condition is \(C^r \equiv
$RD^n (mod \ w)$. An adversary $\hat{A}$ trying to impersonate BS to some node $j$ can guess the challenge and prepare a commit: $RD^{-n}$ using its guess. $\hat{A}$ would then send $r$ as the response. If $\hat{A}$’s guess was correct, then $j$ would successfully verify the ZK proof, and the extracted value would be: $r.S^{-n}$. Nonetheless, $\hat{A}$ would fail to construct the challenge key that $j$ would derive. This is because, for $\hat{A}$, this key uses the extracted value $r.S^{-n}$, which involves the unknown secret $S$, along with $v$, which is also unknown.

**DH with no public exchanges** Our protocols do not transfer key parts explicitly, yet the recipient node is capable of building a one-time session key with BS to extract the permission secret at no additional overheads except for the extra bits of information sent and the single point multiplication (for ECC-DH). So even if an attacker launches a timing attack on the exchanges, the computing time cannot be related to particular values, since no key parts are sent in the clear. In TL3, we relax our requirements slightly by assuming that one of the key parts is known, nonetheless, the other key part involving $g$ is never communicated in the clear, again elevating the security of the scheme.

**Collusion resistance** We define collusion resistance as follows: For all nodes $j$ in $W$, the compromise of $j$ does not affect the network’s ability to bootstrap any other node $i$.

For example, if BS shares the same key with all nodes in $W$, then the compromise of a single node directly affects the network’s ability to bootstrap any other node, since the compromised node can easily learn the communicated secrets. On the other hand, if nodes share unique secrets with BS, but symmetric key or public key primitives are used, then if those keys were not periodically updated, they can degrade over time, and a compromised node which gets actively involved in intercepting the messages sent to other nodes could successfully compromise the nodes’ secrets.
Theorem 7.4  *The bootstrap approach is collusion resistant.*

**Proof**  For each node $j$, $j$ shares unique main secrets with BS, that are used in a zero knowledge way. Based on the assumption that keys used in ZK schemes do not degrade over time, along with Theorem 3.1 and Theorem 3.2, we can conclude that no nodes other than $j$ can learn any of $j$’s main secrets during the bootstrap of $j$, so these secrets can be used indefinitely to bootstrap $j$ with new group secrets.

**Scalability**  Our bootstrap approach is made scalable by allowing BS to delegate trust to a selected set of nodes in group $G$, which in turn act as a trusted base (TB) for that group. The delegation of trust can be achieved by communicating the permission secret $g$ to a selected set of nodes forming the TB of $G$, following which, BS computes and maintains the key part $k_g$ in terms of $g$ while permanently deleting $g$. BS can then use $k_g$ to grant permissions to $G$ that are verifiable only by TB of G. Once a node’s membership has been verified by a TB node, selected security materials are communicated to this node, consecutively, the node becomes a member of the subgroup of $G$ holding those secrets. It should be noted that if $G$ spans the whole network, the trusted base of $G$ can be viewed as a set of base stations, each of which are capable of bootstrapping any node $j$ in the network. This hierarchical delegation of trust enables the scalability of the proposed bootstrap approach while emphasizing the independence of the trust levels.
Chapter 8: Discussions and Conclusions

In this chapter I summarize our contributions and work conclusions and present future directions of extending the work.

8.1 Summary and Conclusions

I have proposed a new interactive proof system and a new model that uses the notion of a common language between two parties Peggy and Victor. Unlike traditional formulations of interactive proofs, I assume that Victor is as knowledgeable as Peggy on a common set of inputs constituting their common language. An interactive proof then establishes that an input accessible to both Peggy and Victor is in a common language of the two.

Traditionally, Victor verified a proof of knowledge or computational ability of Peggy on an input $I$. In the original model, an honest Peggy had knowledge of $I$, while Victor had no knowledge of its own, followed its protocol and relied solely on its random tape in formulating its messages. Knowledge at the verifier was considered only in the context of modeling adversarial verifier strategies where the proof systems had to be secure against any possible verifier regardless of the knowledge available to this verifier.

In contrast, our work models both honest Peggy and Victor as knowledgeable parties, and considers ways in which the knowledge at Victor can be used to improve the security properties of the proof systems. I consider forms of knowledge at Victor that can be related
to the knowledge of Peggy, and show that this relation can be used to construct a common language of both.

My proofs on common languages guarantee that no other assertion is leaked beyond the assertion that both Peggy and Victor have knowledge. In case the assertion is false, nothing should be learned beyond that either, i.e., if an adversary impersonating Victor interacts with Peggy on $I$, he should learn nothing about the state of knowledge of Peggy. We have shown that proofs on common languages can not be constructed using proofs of knowledge or proofs on computational ability.

Using proofs on common languages for authentication allowed different parties to authenticate themselves on a basis other than identities. Instead of identity based trust, we proposed establishing trust based on common languages which maintains a party’s privacy and anonymity.

I demonstrated that zero knowledge variants of proofs on common languages constitute efficient secure two party computations that is not based on Yao’s garbled circuits. We also proposed to model the authentication of Peggy and Victor as a computable function, and have shown that our approach using proofs on common languages guarantees secure computation of this authentication function. Our approach is resilient to server breaches without requiring multiple servers, contrast to known approaches of multiparty secure computations in the context of authentication.

My work considered symmetric as well as asymmetric knowledge for Peggy and Victor, I also considered protocols in which Victor sends only random coins, and others in which Victor uses its knowledge or computational ability along with its coin tosses to formulate its messages to Peggy. I proposed common languages and relations for a number of NP languages, including Discrete Log, RSA, and Graph Hamiltonian, and demonstrated
the possibility of designing common languages using other common languages through a common language proposal for the graph non isomorphism problem, which is in co-NP. I then proposed protocols realizing the proposed language and relation formulations. My protocols demonstrate the applicability and practicality of proofs on common languages for symmetric or asymmetric cryptographic systems.

I proposed a number of applications and realizations for the use of common language based trust for Authenticated Key Exchange (AKE), secret handshake, federated anonymity, and delegation. My proposals considered applications for both the symmetric and asymmetric cases for the knowledge of Peggy and Victor.

One important application that I especially emphasized is the use of common languages for secrets that need not be updated to maintain security, I denoted these as "lifetime secrets". I motivate Lifetime keys particularly for device networks where symmetric cryptography prevails and solutions are required to reduce key management overhead.

I also considered using the common language of Peggy and Victor in the context of existing proof systems, and proposed practical and efficient extensions to these proof systems that made use of the shared knowledge of Peggy and Victor to improve soundness without using multiple rounds. The soundness improvement was based on the use of Victor’s knowledge to extract and verify the correctness of the randoms in Peggy’s zero knowledge (ZK) proofs. I demonstrated an application of this model for bootstrapping device networks in the field. My solution is minimal in the sense that it circumvents ongoing maintenance of security material. Specifically, security material is communicated to members of a device group just once by using zero knowledge identification over a common language established by the preshared secrets. Devices in the group may henceforth securely verify each
other as well as initialize mutual keys for confidentiality without needing to update that security material over time.

8.2 Discussion

I believe that our work can be extended in many ways, and that there are still a number opportunities in exploring many related concepts. In this section I propose and discuss different directions for extending the work for proofs on common languages.

8.2.1 A Universal Construction for Common Languages

Central to the design of protocols for proofs on common languages is the design of the common language and its relation. If it is possible to derive a general construction for each class of languages, the design of these protocols would be highly simplified. We believe that we do have a general approach for NP languages, and co-NP languages where we use solvers for NP problems as proofs of membership in the common language. Nonetheless, other components of the common language model need to be accurately specified for identical and non identical provers as well.

8.2.2 Private Versus Public Coin

Goldwasser and Sipser have shown that any private coin protocol \(\langle P, V \rangle\) for a language \(L\) can be transformed into a public coin protocol \(\langle P', V' \rangle\) for \(L\) with the same round complexity. As stated in [61], from a cryptographic perspective, the transformation in [45] does not preserve the efficiency of the prover and can thus not be applied to computationally sound protocols (arguments) or properties such as zero knowledge as it requires the prover to run in super-polynomial time [61].
Our protocols make assumptions on the efficiency of the prover, and its security properties (namely, zero-knowledge and soundness), rely mainly on the hardness of inverting one-way functions. In the literature we can find similar cases, for example, the protocol of proving Graphs non isomorphism, whose security cannot be preserved in the public coins model.

It should be noted that class AM (public coins) is a subset of IP (which contains public and private coins), thus, some languages may not admit public coins solutions. Nonetheless, our new proof model based on Common Languages does not admit the usual transformation (Goldwasser and Sipser’s transformation from private to public coins).

On the other hand, the transformation in [61] seems like a valid possibility for proofs on common languages. This transformation is from private-coin protocols into Sam-relativized public-coin protocols; where Sam is a collision-finding oracle. The transformation preserves zero knowledge, round complexity and communication complexity. Nonetheless, it focuses on fully black-box protocols based on one-way functions and makes the strong assumption of the use of a collision-finding oracle. Moreover, the transformation will be from class IP to CF and not AM.

The challenge for finding a suitable transformation for proofs on Common Languages is inherent in the use of “common” instead of “a” language. Interactive proofs are the probabilistic analogous to NP proofs. For NP proofs the prover produces a full proof that is read/verified by the verifier. In this model, the verifier does not actively participate in the proof process or interact with the prover in any way. It suffices for the prover to speak and the verifier to listen [45].
Our model adopts the same requirement for the passive involvement of the verifier, or in our model, party \( Y \), which performs the verification. In order to simplify the Zero Knowledge requirements. Another important reason is that we would like our proofs to continue being a probabilistic analogous to NP proofs (efficiently verifiable proofs). Nonetheless, in our model, the verifier needs to do active listening. It needs to decode the proof before verifying it, using the common language. This decoding implies that the proof should be actively encoded by the help of the verifier, hence the use of private coins.

We believe that the transformation in [61] can be adopted for proofs on common languages with slight modifications. On the other hand finding a transformation that does not make use of nonstandard assumptions remains a possible future extension of our work.

The implication of public coin proofs on Common Languages is comparable to those of the standard model, which favors public coin to private coin. To exemplify this, consider leakage-resilience ZK. This property requires the use of public coins to achieve full leakage resilience.

### 8.2.3 Resettable Zero Knowledge

A protocol that is resettable ZK remains zero knowledge even if an adversary can interact with the prover many times, each time resetting the prover to its initial state and forcing it to use the same random tape. All known examples of zero knowledge proofs and arguments are trivially breakable in this setting. Moreover, by definition, all zero-knowledge proofs of knowledge are breakable in this setting [24].

In all the protocols that we’ve proposed as part of our common languages protocol suite, party \( Y \) is the one that starts the interaction messages. In each of these protocols \( Y \) sends only one message, and it never sends enough information that allows its verification at \( X \),
and in some protocols, such as CDLL, Y even relies totally on its random coins to formulate its message to X. The party whose messages have the potential to leak knowledge is always X. Nonetheless, in all our protocols, party X always replies to Y’s message. I.e., Y goes first, followed by X. This makes it hard to force X to reuse its randoms through resetting, as X can always base its random selection on the message it receives from Y. This is implemented through the use of the \textit{rand} function in most of our protocols. A further study of resettable ZK in the context of common languages is needed to formally prove the above argument.

\subsection{A Stronger Notion of Zero Knowledge}

The common language model can allow for stronger notion of zero knowledge, particularly one that guarantees that even if an adversary succeeds in extracting any knowledge from a given proof session, particularly the random coins used in the proof interaction, the reuse of the leaked knowledge in another session will not give this adversary any advantage in terms of deriving the secret(s). I claim that protocols satisfying this new notion are ZK regardless of the oracle model used, i.e., random versus standard oracle model. In essence, this is different than resettable ZK (rZK), as an rZK protocol remains zero knowledge even if an adversary can interact with the prover many times, each time resetting the prover to its initial state and forcing it to reuse its randoms.

Similarly it is also different than leakage resilient ZK (LR-ZK), where an(LR-ZK) protocol does not yield anything beyond the validity of the assertion and the leakage obtained by the adversary, i.e., whatever an adversary learns from the protocol (with leakage) should be no more than what it can learn from only the leakage without running the protocol. The notion of leakage used in LR-ZK includes any leakage on the prover’s state, i.e., it may
even be a leakage on the witness itself. On the other hand, the suggested ZK notion focuses on leakage during the protocol execution, and is concerned specifically with leakage on the random coins of the parties involved. We consider using a leakage query oracle similar to the oracle used for formalizing LR-ZK. Nonetheless, the ZK simulator use of the oracle will be different than the oracle use for LR-ZK.

8.2.5 Practical Evaluation of Proofs on Common Languages

The common perception of public key cryptography is that it is complex, slow and power hungry, and as such not suitable for resource constrained Device Networks. Hence, symmetric key protocols have been widely used for such networks. Nonetheless, overtime, such protocols leak knowledge about the underlying symmetric secrets, which mandates costly periodic key updates.

Generally, ZK protocols are considered lightweight public key cryptography (originally ZK protocols have been introduced for smart cards). We proposed ZK protocols for proofs on common languages that use symmetric or asymmetric secrets in an efficient way. Our protocols are more efficient than existing ZK protocols as they require fewer messages and less rounds. Such protocols can make use of the existing symmetric secrets infrastructure of Device Networks, or introduce asymmetric secrets and ensure that these secrets will be lifetime, i.e., persistent and will not degrade over time. It then follows that minimal key management operations are required. A comparative practical evaluation of these protocols against existing symmetric protocols for authentication in Device Networks would be relevant to justifying the use of our methods in practice.
Bibliography


193


