Security enhanced agent systems

Qi Zhang
University of Wollongong


This paper is posted at Research Online.
NOTE

This online version of the thesis may have different page formatting and pagination from the paper copy held in the University of Wollongong Library.

UNIVERSITY OF WOLLONGONG

COPYRIGHT WARNING

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.
Security Enhanced Agent Systems

A thesis submitted in fulfillment of the requirements for the award of the degree

Master by Research

from

UNIVERSITY OF WOLLONGONG

by

Qi Zhang

School of Computer Science and Software Engineering
November 2009
Dedicated to

my parents ... 
with love and gratitude
Declaration

I, Qi Zhang, declare that this thesis is wholly my own work unless otherwise referenced or acknowledged below. The document has not been submitted for qualifications at any other academic institution.

Qi Zhang
November 18, 2009
Certification to Thesis

Examination Committee

I (with my co-authors) wrote three research papers (as listed on page ix) during my study. The contents of these papers were used in my thesis. I declare that I have made the major contribution to these papers. As co-authors, Associate Professors Yi Mu and Minjie Zhang were also my thesis supervisors and Professor Robert H Deng is my external advisors. They provided their professional advices and guidance to me during my study. These advices and guidance were indeed very helpful and led to the success of our study. I would like to declare that I have contributed around 80% of the work, which captures the major development of the schemes and security proofs.

Qi Zhang
November 18, 2009
Abstract

Software agents are useful for distributed systems and electronic commerce. However, to fully deploy software agents in practice, a number of challenging issues, especially security and privacy, need to be addressed. In general, software agents can be classified into mobile agent and multi-agent, which have different security requirements.

Mobile agents are mobile in the sense that they can move in the defined computer network. Due to this nature, security and privacy become critical. When a mobile agent travels in a hostile environment or migrates to an untrusted platform, its security and privacy can be easily compromised. In particular, the remote hosts in which agents visit and get services are not considered to be trusted. Existing solutions suggest that remote hosts together with the agent’s home jointly sign the service agreement. Therefore, proxy-based signing model was utilized. We observe that this actually poses a serious problem: a host which should be excluded from the desirable hosts could also generate a signed service agreement. In order to solve this problem, we propose a secure mobile agent transaction scheme which achieves host authentication with designated hosts. In our scheme, only selected hosts can be included in the agent network and hence generate a valid signed service agreement. We also propose a variant of our scheme that provides a shorter signature size.

Multi-agent systems are different from mobile agents systems in that they are not mobile. Although multi-agent systems do not have the security risks stemming from mobility, they have other security problems. Unfortunately, security and privacy issues have not been adequately addressed. Most proposed schemes only concern with security protection rather than privacy protection. Privacy issues have not drawn adequate attention and actually been ignored or mistreated in most proposed multi-agent protocols. We argue that privacy issues are indeed not trivial and cannot be resolved with traditional security mechanisms. If agents do not trust each other, their privacy must be protected. In order to solve the issue, we
propose a novel secure multi-agent protocol which captures several most important
security properties including data confidentiality, agent privacy and authenticity.
In our scheme, privacy protection is applied to both negotiating parties (agents).
The security protection in our scheme satisfies the most stringent security level, i.e.,
indistinguishability against adaptive chosen ciphertext attacks.
Acknowledgement

First of all, I am very grateful to my supervisor, A/Prof. Yi Mu, who gave me this opportunity to study cryptography. He was always very patient to answer my questions and provide professional guidances. I also would like to express my gratefulness to my co-supervisor, A/Prof. Minjie Zhang, who gave me her professional advices on improving my knowledge of software agents.

Another great gratitude goes to my parents. When I encounter problems, their encouragement always helps me to move forward.

I am very grateful to my beloved girlfriend Yidan, for her love and sacrifice. During my thesis writing, she took most of the housework to make sure I can concentrate on my thesis.

I would like to express my appreciation to Dr. Xinyi Huang. It is a pleasure to discuss research with him. I appreciate his clear explanations on some cryptographic topics and critical comments on my papers. I would like to thank Ms Wei Wu, for her time and patient discussions on the research topics.

I thank the administrative staff and the technical support staff of School of Computer Science and Software Engineering, Faculty of Informatics, for their supports.

Qi Zhang
Wollongong, August 2009


Qi Zhang, Yi Mu, and Minjie Zhang. Secure mobile agents with less communication overhead. (draft)
## Contents

Certification to Thesis Examination Committee v

Abstract vi

Acknowledgement viii

Publications ix

1 Introduction 1
   1.1 Security Issues in the World of Agents ......................... 1
   1.2 Existing Works ........................................... 3
   1.3 Challenging Issues and Motivations .......................... 3
   1.4 Proposed Solutions ......................................... 5
   1.5 Overview of the Thesis ..................................... 6

2 Background 7
   2.1 Preliminaries ................................................ 7
      2.1.1 Miscellaneous Notations ............................... 7
      2.1.2 Random Oracle Model .................................. 8
      2.1.3 Polynomial Time (Complexity Theory) .................. 8
      2.1.4 Bilinear Map ........................................... 8
   2.2 Number-Theoretic Problems .................................... 9
      2.2.1 The Discrete Logarithm Problem ........................ 9
      2.2.2 The Diffie-Hellman Problem ............................ 10
      2.2.3 The Bilinear Diffie-Hellman Problem ................. 10
      2.2.4 The q-Strong Diffie-Hellman Problem ............... 11
   2.3 Hash Function .............................................. 11
   2.4 Public Key Encryption ........................................ 12
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4.1 Identity-Based Encryption</td>
<td>13</td>
</tr>
<tr>
<td>2.5 Digital Signatures</td>
<td>14</td>
</tr>
<tr>
<td>2.6 Proxy Signatures</td>
<td>15</td>
</tr>
<tr>
<td>2.7 Accumulators</td>
<td>17</td>
</tr>
<tr>
<td>2.8 Secret Handshakes</td>
<td>18</td>
</tr>
<tr>
<td>2.9 Oblivious Signature-Based Envelope</td>
<td>19</td>
</tr>
<tr>
<td>2.10 Proxy-Based Mobile Agents</td>
<td>20</td>
</tr>
<tr>
<td>2.11 Multi-Agents</td>
<td>24</td>
</tr>
<tr>
<td>3 Secure Mobile Agents with Designated Hosts</td>
<td>25</td>
</tr>
<tr>
<td>3.1 Task Execution Procedure of Proposed Mobile Agent</td>
<td>26</td>
</tr>
<tr>
<td>3.2 Security Model</td>
<td>27</td>
</tr>
<tr>
<td>3.2.1 Existential Unforgeability Against Malicious Hosts</td>
<td>27</td>
</tr>
<tr>
<td>3.2.2 Existential Unforgeability Against Malicious Customer</td>
<td>29</td>
</tr>
<tr>
<td>3.2.3 Complexity Definitions</td>
<td>30</td>
</tr>
<tr>
<td>3.3 Our Scheme</td>
<td>30</td>
</tr>
<tr>
<td>3.4 Security Analysis</td>
<td>32</td>
</tr>
<tr>
<td>3.4.1 Existential Unforgeable Against Adversary $A_1$</td>
<td>32</td>
</tr>
<tr>
<td>3.4.2 Existential Unforgeable Against Adversary $A_2$</td>
<td>37</td>
</tr>
<tr>
<td>3.4.3 Existential Unforgeable Against Adversary $A_3$</td>
<td>39</td>
</tr>
<tr>
<td>3.5 Summary</td>
<td>40</td>
</tr>
<tr>
<td>4 Secure Mobile Agents with Less Communication Overhead</td>
<td>41</td>
</tr>
<tr>
<td>4.1 Security Model</td>
<td>41</td>
</tr>
<tr>
<td>4.1.1 Complexity Assumptions</td>
<td>42</td>
</tr>
<tr>
<td>4.2 Our Scheme</td>
<td>42</td>
</tr>
<tr>
<td>4.3 Security Analysis</td>
<td>44</td>
</tr>
<tr>
<td>4.3.1 Existential Unforgeable Against Malicious Host within Designated List</td>
<td>44</td>
</tr>
<tr>
<td>4.3.2 Existential Unforgeable Against Malicious Host without in Designated List</td>
<td>48</td>
</tr>
<tr>
<td>4.3.3 Existential Unforgeable Against Malicious Customer</td>
<td>49</td>
</tr>
<tr>
<td>4.4 Comparison</td>
<td>49</td>
</tr>
<tr>
<td>4.5 Summary</td>
<td>50</td>
</tr>
</tbody>
</table>
List of Tables

4.1 Bandwidth Comparison Between Two Schemes. . . . . . . . . . . . . 49
4.2 Performance Comparison Between Two Schemes. . . . . . . . . . . . 49
List of Figures

3.1 Agent-based Online Application Architecture . . . . . . . . . . . . . . 27
5.1 Attribute-based Information Exchange in the Multi-agent System . . 52
Chapter 1

Introduction

A software agent is a piece of software code relying on a host platform for execution [Nwa96, Mil97]. Every software agent has an owner. Normally an owner delegates tasks associated with part of its privilege to an agent so that the agent is able to implement certain tasks on behalf of the owner. The most interesting property is that a software agent is capable of flexible and autonomous actions. In other words, the agent is able to implement the task without interference from the owner. There are advantages and disadvantages, which exposes challenging issues especially in the security area.

1.1 Security Issues in the World of Agents

In general, software agents can be divided into several categories which include intelligent agents, distributed agents, multi-agents and mobile agents [Woo02, JW98, JSW98, PW04]. They all have different special capabilities suited for different application domains [ZM02]. Mobile agents should be distinguished from the other agents, for example, intelligent agents, distributed agents, and multi-agents, due to the mobility they possess. The other agents can fall into the category of fixed agents which do not possess mobility. The focus of this thesis is on the security of mobile agents and multi-agents.

A mobile agent is able to travel across the network performing tasks on behalf of its owner. Behaviours of a mobile agent are fully autonomous without interference from its owner after being launched from its home platform. The mobile agent technology has drawn much attention in recent years because of its potential to bring new ways in electronic commerce [KBC00, SL03]. As an example, a mobile agent could be released by its owner to get the best deal from one of many online sellers. It can travel around the network to search and negotiate with the suitable
1.1. Security Issues in the World of Agents

sellers. After the deal is done, it returns the result to its owner.

Although it is generally believed that mobile agents are a powerful utility for electronic commerce, security and privacy is indeed a concern. As a mobile agent is in the form of a piece of software code, it can not perform any task without a host. Major issues include their inability to authenticate transactions in hostile environments [KBC00] and the difficulty of hiding any private information from being known to the outside, for example, malicious host and third parties. When a mobile agent arrives at a remote host, it will be fully controlled by the host. Therefore, it is believed that it is impossible for a mobile agent to carry out any secret computation without exposing its secret to the malicious host [Che98, ACCK01].

Multi-agent is another significant sub-category in the world of software agents. In general, a multi-agent system consists of a group of agents associated with a trusted authority which is in charge of managing the group. The trusted authority manages the group by, for example, forming or disbanding the group and managing memberships. Multi-agent systems have been implemented broadly into distributed systems because they are more capable of solving a complex problem, in comparison with a sole agent system [GCL+05].

In many Internet-based applications such as online trading, the multi-agent system is formed by unacquainted agents who came from different owners. Therefore, confidentiality and authenticity amongst the agents in a group is critical in order to allow the agent to function securely in the system [HJS06, SD06, ASDB08, FB07]. Consider a scenario, a multi-agent system where the agents are grouped into different groups. Each group is managed by a trusted group manager. In practice, this could be a situation in electronic commerce, where agents are sellers and buyers who are registered with a group. At the seller’s side, a seller does not want to reveal its offer to the other sellers in order to protect its commercial advantage and vice versa for the buyer. On the other hand, for privacy reasons, both sellers and buyers also do not want to reveal their affiliations (e.g. seller, buyer) to unexpected parties when offers are exchanged. This common scenario actually raises a critical security issue: how can the agents exchange messages safely and correctly, if no party wants to reveal its affiliation to the other parties first? In other words, how can an agent make sure that its messages are only visible to the parties with desired affiliations before their reputation is built?
1.2 Existing Works

One of the earliest methods to mobile agents security was introduced by Sander and Tschudin [ST98]. They concluded the following fundamental problems of executing mobile code: code and execution integrity, code privacy, and computation with a secret in public. They gave an answer to the above problems by proposing a concept called Computing with Encrypted Functions (CEF). Kotzanikolaou et al [KBC00] implemented the CEF scheme and proposed an undetachable signature scheme based on RSA. Although their scheme could conceal the agent owner’s private key during an execution in an untrusted environment, it does not provide the fairness of contract [LKK01b], since the remote host is not committed to the transaction whereas the customer is. Therefore, the commitment of the host is required in the transaction to prevent an impersonation attack to the host.

One of the most important security services to mobile agent systems is non-repudiation, which provides fairness of transactions to hosts and agent owners. Proxy signature is thought to be an appropriate solution to the repudiation issue in mobile agent applications. We notice that several solutions derived from proxy signatures were published [SL03, LKK01b, KBLK01, PL01, LKPY06, LKK01a, HMS+05, WMS+07]. Unfortunately, many of them are insecure [PL01, LCK03, LKK01b, WBZD03]. By default, a proxy-based approach such as Lee et al [LKK01b] grants the universal signing privilege to all hosts in the agent network; that is, any host can generate a valid signature by executing a mobile agent. In other words, a non-repudiation service agreement between the agent owner and any host can be reached out of the control of the agent owner. This assumption would be fine, if the agent owner wants to receive services from all hosts in the agent network including the undesirable ones.

In the area of multi-agent systems, most existing solutions focus only on the security protection of the messages exchanged among the agents. Privacy protection of the agents themselves has not drawn much attention and has actually been ignored.

1.3 Challenging Issues and Motivations

Existing solutions for mobile agents do not offer the feature of designated hosts. Therefore, any host in the agent network has the capability of generating a valid
signed service. This actually poses a serious problem in that an agent owner might not regard that all the hosts in the agent network are desirable for a designated service. For instance, the agent owner wants to get services such as mortgage information from some specified banks only. Therefore, hosts that do not belong to these banks should be excluded from the network. We argue that the feature of designated hosts is a significant feature among mobile agents since it is required by many practical applications. We found that there is no such a solution equipped with this feature.

Most existing solutions of multi-agent systems only provide security protection on messages but not on privacy. We argue that protecting the messages is not sufficient, since the privacy protection is important as well in many applications such as Internet-based e-commerce and e-auction. Therefore, it is necessary to build a scheme including both security and privacy protection for multi-agent systems. We are motivated by two cryptographic primitives: oblivious signature-based envelope (OSBE) [LDB03a, NT05] and hidden credentials [HBSO03, BHS04] to build such a scheme.

OSBE is a secure information exchange protocol with authentication enabled. It supports interactive authentication where only the counterparts, which are authenticated successfully, are able to decrypt the ciphertext sent by the sender. Actually, Boneh and Franklin’s [BF01] identity-based encryption (IBE) can be regarded as a special case of OSBE where it utilizes the identity as the policy attribute and the private key is a short signature to the policy attribute. With the OSBE, a sender can communicate with a receiver such that the receiver’s role (or policy attribute) is unknown to the sender. The receiver can decrypt the ciphertext sent by the sender, only if the attributes he holds matches the sender’s policy attributes. At the first glance, we believed that the OSBE could be a suitable candidate for our scenario. Unfortunately, with a closer look at the OSBE scheme, we found that it lacks fairness in authentication between the agent members. The sender is not authenticated by the receiver during the negotiation phase. Therefore, any one including the malicious sender can send an encrypted message to a receiver. The receiver receives no knowledge about the sender after the negotiation completed. Hidden credential protocols, which are analogous to OSBE, offer an additional feature of hiding the policy attribute from disclosure. Unfortunately, the paper [HBSO03] does not present a concrete scheme to show how it works. To achieve security and privacy protection simultaneously is indeed not trivial, since there is no existing secure algorithm that
provides this property.

1.4 Proposed Solutions

In response to the challenging issues mentioned previously, we propose three schemes, which form the main contributions of this thesis.

We propose a novel secure mobile agent scheme which allows designated hosts to perform an agent task. The list of designated hosts can be chosen by the agent owner. The scheme can be regarded as a policy-based scheme, in that the agent can carry a tamper-resistant policy data set when implementing a task, where the hosts defined in the policy are included in the specific network and can provide services. One of the features in our scheme is that the policy can be dynamically updated for various tasks without any additional computation cost. This feature inherits the elegant property from the dynamic accumulator [Ngu05, GTH02, CL02], in that it does not increase the size of policy data set when the policy is changed. The scheme also inherits the merit from proxy-based approaches that the repudiation issue is eliminated. Details of this scheme are presented in Chapter 3.

Further more, we propose a novel mobile agent scheme derived from the scheme presented in Chapter 3. Compared to other mobile agent schemes, the signature length of our scheme is the shortest. Our scheme is equipped with the feature of designated verifier [JKI96, LV04, SKM03, SLBP04, SZM04]. In other words, only the designated verifiers are able to verify the validity of a signed deal which is jointly signed by the agent owner and the remote host [Wan04, SLBP03]. It is a useful feature especially in the situation that the privacy of agreement must be protected. Details of this scheme are presented in Chapter 4.

For solving the security issues in the multi-agent systems, we propose a secure information exchange scheme with authentication enabled. Our scheme provides not only a security protection to the information exchanged among agents but also a privacy protection to the negotiating agents. An attribute-based authentication mechanism has been applied into our scheme to protect the privacy of the agents. Each authentication can be successful if and only if both the negotiating agents hold the correct attributes. The messages can be exchanged successfully among the agents only if the authentication is successful. In case of authentication failure, no information will be revealed to the other parties. Our scheme can be very useful in the case, for example, Internet-based e-auction, where the privacy (e.g. affiliations)
of the negotiation parties must be protected [BPJ02, KB01, Kra01, JFL+01, FSJ98, FWJ01]. Details of this scheme are presented in Chapter 5.

1.5 Overview of the Thesis

This thesis concerns resolving the security challenges in the area of mobile agents and multi-agents. The rest of this thesis is organized as follows:

Chapter 2 introduces the background knowledge of cryptography and the mathematical hard problems, which include the concept of bilinear maps, the topics of public key cryptography, and a brief introduction to hash functions. The mathematical problems we present here are the security fundamentals of our proposed schemes.

In Chapter 3, a secure mobile agent scheme for online transactions is introduced. We also present the comprehensive definitions of the security models along with rigorous security proofs.

In Chapter 4, a novel mobile agent scheme is proposed. This scheme is derived from the secure mobile agent scheme presented in Chapter 3. Compared with the scheme in Chapter 3, the size of the mobile agents in this scheme is shorter. We also present the definition of the security model together with the security proofs to this scheme. A comparison of bandwidth and performance between this scheme and the scheme presented in Chapter 3 is provided.

In Chapter 5, we define a general architecture of our multi-agent system and several security properties. Then, a novel secure information exchange scheme for multi-agent systems is proposed. Associated security models and proofs are also provided.

In Chapter 6, a summary is presented.
This chapter provides an introduction to the topics of number theory and cryptography. These topics will be used in the subsequent chapters. Readers who are familiar with these topics may skip this chapter.

### 2.1 Preliminaries

#### 2.1.1 Miscellaneous Notations

We denote by ID the identity of an individual, such as a remote host, or an agent. An ID can be represented by an arbitrary string such as name, email address, and IP address. For example, "bob", "bob@company.com" or "58.121.192.1" can be used as an ID of an individual. As the ID is used to distinguish the individuals, no two individuals share the same ID.

Let $\hat{e}$ denote a function that maps a pair of mathematical group elements to a mathematical group element. We denote by $\hat{e}$ a bilinear map. The mathematical group is a nonempty set together with an associative binary operation. The definition of bilinear map is presented in Section 2.1.4.

The expression $x \leftarrow Y$ means that an element $x$ is chosen randomly from a finite set $Y$ with uniform distribution.

The symbol $|$ denotes the concatenation of binary strings. For example, $a||b||c$ represents a binary string concatenated by three strings $a$, $b$, and $c$.

An expression in conjunctive normal form (CNF) is a conjunction of terms where each term is a disjunction of strings. For example, $(A \lor B) \land C$ is in CNF, where $A, B, C \in \{0, 1\}^*$ are arbitrary strings. An expression in disjunctive normal form (DNF) is a disjunction of terms where each term is a conjunction of strings. For example, $(D \land E) \lor F$ is in DNF, where $D, E, F \in \{0, 1\}^*$ are arbitrary strings. We
use an expression in CNF (or DNF) to describe the combination of policy attributes held by the agents. For example, the expression \((D \land E) \lor F\) denotes a policy which requires an agent to possess either attribute \(D\) and \(E\), or attribute \(F\).

### 2.1.2 Random Oracle Model

In many cryptosystems, cryptographic hash functions (the definition of hash function is presented in Section 2.3) are often required. An ideal cryptographic hash function is a mathematical function which possesses the special properties such as one-wayness and strong randomness. However, there are no such ideal functions in the real world. Therefore, the random oracle model was introduced. A random oracle is a mathematical abstract function which takes as input each query and responds with a random choice from the output domain [BR93]. For the same queries, the random oracle responds with the same output. A random oracle is used to replace the ideal hash functions in security proofs. This kind of security proof is referred to as the random oracle model.

### 2.1.3 Polynomial Time (Complexity Theory)

An algorithm is a finite computational procedure which starts with some inputs and terminates with an output for solving a problem [CLR92]. Generally, the efficiency of an algorithm is evaluated by considering the resources, i.e., running time (computation steps) and memory, required to solve the given problem [Pap94].

**Definition 2.1** An algorithm is said to be polynomial time if its running time is upper bounded by a polynomial expression in the size of the input for the algorithm.

### 2.1.4 Bilinear Map

A bilinear map is a mathematical abstract function which takes a pair of group elements as input and outputs an element of another group. Normally, we use algebra groups such as cyclic additive groups or cyclic multiplicative groups as the groups defined in a bilinear map.

**Definition 2.2** Let \(\mathbb{G}_1\) and \(\mathbb{G}_2\) be cyclic multiplicative groups of prime order \(q\). Let \(g\) be a generator of \(\mathbb{G}_1\). A function \(\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2\) is a bilinear map if it satisfies the following properties:
2.2. Number-Theoretic Problems

- **Bilinearity**: for all $A, B \in \mathbb{G}_1$ and $x, y \in \mathbb{Z}_q$, $\hat{e}(A^x, B^y) = \hat{e}(A, B)^{xy}$.

- **Non-degeneracy**: $\hat{e}(g, g) \neq 1_{\mathbb{G}_2}$.

- **Computability**: $\hat{e}(A, B)$ is efficiently computable (i.e. in polynomial time) for all $A, B \in \mathbb{G}_1$.

2.2 Number-Theoretic Problems

In modern cryptography, the security of most cryptosystems relies on some intractable mathematical problems. We say a mathematical problem is intractable if no one has found an efficient (in polynomial time) algorithm to solve it. However, there is no proof proposed either such that it is impossible to find an efficient algorithm to solve an intractable problem. Some of the intractable problems are notable such as factoring a large integer and discrete logarithm problem. Hereafter, we describe only the intractable problems which will be used in the following chapter of this thesis. For more information about the intractable problems the readers may refer to [MvOV97]

2.2.1 The Discrete Logarithm Problem

In abstract algebra, the discrete logarithm [McC90, CEdG88] is an operation in a group that is analogous to the ordinary logarithm of real numbers. As an example, let $\mathbb{G}$ be a cyclic multiplicative group and $g$ is a generator of $\mathbb{G}$. For an element $a \in \mathbb{G}$, denote $\log_g(a)$ the discrete logarithm of $a$, which is an unique integer $x$ ($0 \leq x \leq |\mathbb{G}|$) satisfying $a = g^x$.

**Definition 2.3** Given a cyclic group $\mathbb{G}$ and its generator $g$, for any element $a \in \mathbb{G}$, the discrete logarithm problem is to find an integer $x$ where $0 \leq x \leq |\mathbb{G}|$ satisfying $a = g^x$.

The discrete logarithm problem is considered as an intractable problem in mathematics and cryptography [COS86, CS03] because no efficient (in polynomial time) algorithm which can solve this problem has been invented.
2.2.2 The Diffie-Hellman Problem

The Diffie-Hellman problem was firstly proposed by Whitfield Diffie and Martin Hellman [DH76]. It is widely considered as an intractable problem in cryptography. The Diffie-Hellman problem is closely related to the discrete logarithm problem. It is generally believed that the Diffie-Hellman problem can be solved in polynomial time if the discrete logarithm problem can be solved in polynomial time [Mau94, dB90].

There are many variants of the Diffie-Hellman problem. One notable variant is called the Decisional Diffie-Hellman (DDH) problem [JN]. The Diffie-Hellman problem is usually referred to the Computational Diffie-Hellman (CDH) problem in order to distinguish with the DDH problem.

Definition 2.4 Given a cyclic group $G$ over a finite field, a generator $g$ of $G$, and two elements $g^x, g^y$ of $G$ where $x, y$ are two unknown integers, the Computational Diffie-Hellman (CDH) problem is to find the element $g^{xy}$.

Definition 2.5 Given a cyclic group $G$ over a finite field, a generator $g$ of $G$, and three elements $g^x, g^y, A$ of $G$ where $x, y$ are two unknown integers, the Decisional Diffie-Hellman (DDH) problem is to decide whether $A$ equals to $g^{xy}$.

It is obvious that if there exists an algorithm which can solve the CDH problem in polynomial time, then the DDH problem can also be solved in polynomial time.

2.2.3 The Bilinear Diffie-Hellman Problem

The Bilinear Diffie-Hellman (BDH) problems, including the Decisional Bilinear Diffie-Hellman (DBDH) problem and the Gap Bilinear Diffie-Hellman (GBDH) problem, are variants of the Diffie-Hellman problem over the bilinear map. These problems are widely believed to be computationally hard. The security of some cryptographic schemes is based on the BDH such as in [BF01] and [Jou00]. The DBDH was introduced in [BB04a], and the GBDH was introduced in [CC03, OP01].

Definition 2.6 Given cyclic groups $G_1, G_2$ over a finite field, a generator $g$ of $G_1$, a bilinear map $\hat{e} : G_1 \times G_1 \rightarrow G_2$, and three elements $g^x, g^y, g^z$ of $G_1$ where $x, y, z$ are three unknown integers, the Bilinear Diffie-Hellman (BDH) problem is to find the element $\hat{e}(g, g)^{xyz}$ of $G_2$. 
Definition 2.7 Given cyclic groups $G_1, G_2$ over a finite field, a generator $g$ of $G_1$, a bilinear map $\hat{e} : G_1 \times G_1 \to G_2$, three elements $g^x, g^y, g^z$ of $G_1$ where $x, y, z$ are three unknown integers, and an element $A \in G_2$, the Decisional Bilinear Diffie-Hellman (DBDH) problem is to decide whether $A$ equals to $\hat{e}(g, g)^{xyz}$.

Definition 2.8 Given cyclic groups $G_1, G_2$ over a finite field, a generator $g$ of $G_1$, a bilinear map $\hat{e} : G_1 \times G_1 \to G_2$, and three elements $g^x, g^y, g^z$ of $G_1$ where $x, y, z$ are three unknown integers, the Gap Bilinear Diffie-Hellman (GBDH) problem is to find the element $\hat{e}(g, g)^{xyz}$ of $G_2$ with the help of the DBDH oracle.

2.2.4 The q-Strong Diffie-Hellman Problem

The weaker version of the q-Strong Diffie-Hellman (q-SDH) assumption was firstly purposed by Mitsunari, Sakai, and Kasahara for constructing traitor tracing systems [MSK02]. Boneh and Boyen presented the concrete definition of the q-SDH problem together with the proof [BB04b].

Definition 2.9 Given a cyclic group $G_1$ of prime order $p$ over a finite field, a generator $g$ of $G_1$, a $(q + 1)$-tuple of elements $(g, g^x, ..., g^{(x^q)}) \in G_1^{q+1}$ where $x \xleftarrow{\$} \mathbb{Z}_p^*$ is unknown, the q-Strong Diffie-Hellman (q-SDH) problem is to find a pair $(c, g^{1/(x+c)}) \in \mathbb{Z}_p \times G_1$ for a freely chosen value $c \in \mathbb{Z}_p \backslash \{-x\}$.

2.3 Hash Function

In cryptography, hash functions are broadly used for converting arbitrary length messages to fixed length strings which can then be handled by the schemes. Hash functions are also used in the verification phase of many digital signature schemes for achieving data integrity.

Definition 2.10 A Hash function is a mathematical function which maps a set of arbitrary length data to a set of fixed length data. The outputs of hash functions are called hash values.

As an example, a function $H$ which maps binary strings of arbitrary length to binary strings with a fixed length $l$, that is

$$H : \{0,1\}^* \to \{0,1\}^l,$$
can be treated as a hash function.

Naturally, the basic requirement of a hash function is efficiently computable. For cryptographic purposes a hash function $H$ must have the following properties:

- **One wayness:** Given a value $y$, it is hard to find a value $x$ such that $H(x) = y$.

- **Weak collision resistance:** Given a value $x$, it is hard to find a value $x' \neq x$ such that $H(x) = H(x')$ [NY89].

- **Strong collision resistance:** It is hard to find a pair $(x, x')$ where $x \neq x'$ such that $H(x) = H(x')$ [Dam88, BR97].

The hardness of hash functions varies with different setups. The reason is that the hardness of a hash function depends mostly on its output bits, if the best algorithm to break (e.g. find a collision) it is brute force. Numerous cryptographic hash functions have been created so far [And95, Am04, CZK04, LM93]. The notable and broadly-used hash functions include strengthened version of secure hashing algorithm (SHA-1) [oST95], Ron Rivest’s MD5 [Riv92], and the RIPEMD-160/320 [DBP96, DP97]. However, MD5 and RIPEMD-160 have been broken respectively [WY05, WLF+05]. In 2005, a cryptanalysis on SHA-1 reduced the computation complexity of finding collisions from $2^{80}$ to $2^{69}$ [WYY05]. SHA-2 and SHA-3 are suggested to replace the SHA-1 in applications in order to avoid the potential security problems.

## 2.4 Public Key Encryption

Encryption schemes can be classified into two categories: private key or symmetric key encryption schemes and public key or asymmetric key encryption schemes. In a private key encryption scheme, communication parties use the same key for both encryption and decryption. Therefore, key establishment is needed between the communication parties. In a public key encryption scheme, the communication uses a pair of different keys for encryption and decryption respectively. The key used for encryption is referred to as the public key which is published to everyone, while the other key, which is used for the decryption, is referred to as the private key which must be kept secret.

**Definition 2.11** A public key encryption scheme includes three algorithms: $\text{KeyGen}$, $\text{Enc}$ and $\text{Dec}$. $\text{KeyGen}$ is the key pair generation algorithm which takes the system
random parameter $1^k$ as input and outputs a pair of public key and private key,

\[(pk, sk) \leftarrow \text{KeyGen}(1^k),\]

where $pk, sk$ are the public key and the private key respectively. $\text{Enc}$ is the encryption algorithm which takes a message $m$ and the public key $pk$ as input and outputs a ciphertext $c$,

\[c \leftarrow \text{Enc}(m, pk).\]

$\text{Dec}$ is the decryption algorithm which will output the encrypted message $m$ if and only if the input ciphertext $c$ and the private key $sk$ satisfy the following equation,

\[
\text{Dec}(c, sk) \begin{cases} 
  = m, & \text{if } c \leftarrow \text{Enc}(m, pk) \text{ and } (pk, sk) \leftarrow \text{KeyGen}(1^k). \\
  \neq m, & \text{otherwise.}
\end{cases}
\]

The fundamental requirement of a public key encryption scheme is that it is infeasible to compute the private key from the public key together with any other information. In practice, the computation of public key encryption schemes is much slower than that of private key encryption schemes at the equivalent security. Therefore, a reasonable way of encrypting large messages is to encrypt a session key using public key encryption. Once the session key has been established the communication parties can choose a fast symmetric cipher to encrypt messages.

### 2.4.1 Identity-Based Encryption

Identity-based encryption (IBE) is a kind of public key encryption which employs the participant’s unique identity (e.g. email address, ip address) as the public key. A trusted third party, which is called the Private Key Generator (PKG), is introduced in IBE which is in charge of issuing the corresponding private keys to the participants according to their identities (public keys). Initially, the PKG needs to generate a master private key which must be kept secret. For every single participant, the PKG generates the corresponding private key by using the master private key.

The idea of IBE was firstly proposed by Adi Shamir [Sha84] in 1984 together with the implementation that allows the signer’s identity being used as the public key to verify the digital signatures. After that, Dan Boneh and Matthew Franklin [BF01] presented the first implementable IBE scheme from pairings. Another approach for IBE was independently proposed by Clifford Cocks [Coc01], where the security model is based on the quadratic residuosity assumption.
Definition 2.12 An identity-based encryption (IBE) scheme includes three roles: PKG, senders, and receivers; and four algorithms: Setup, KeyGen, Enc, and Dec. Setup is run initially by the PKG to setup a random IBE system environment,

\[(\text{params}, \text{msk}) \leftarrow \text{Setup}(1^k)\],

where params is the system’s parameter and msk is the master private key. KeyGen is the private key generation algorithm run by the PKG which takes the system parameter params, the identity ID and the master private key msk as input and outputs a corresponding private key \(sk_{\text{ID}}\),

\[sk_{\text{ID}} \leftarrow \text{KeyGen}(\text{params}, \text{ID}, \text{msk}).\]

Enc is the encryption algorithm run by the sender which takes the system parameter params, the prospective receiver’s identity ID and the message \(m\) as input and outputs a ciphertext \(c\),

\[c \leftarrow \text{Enc}(\text{params}, \text{ID}, m).\]

Dec is the decryption algorithm which will output the message \(m\) if and only if the input ciphertext \(c\) and the private key \(sk_{\text{ID}}\) satisfy the following equation,

\[
\text{Dec}(\text{params}, c, sk_{\text{ID}}) = \begin{cases} 
= m, & \text{if } c \leftarrow \text{Enc}(\text{params}, \text{ID}, m) \text{ and} \\
& sk_{\text{ID}} \leftarrow \text{KeyGen}(\text{params}, \text{ID}, \text{msk}). \\
\ne m, & \text{otherwise}.
\end{cases}
\]

The most significant characteristic of IBE is that no prior key pairs distribution is needed, which is useful especially when the key pairs pre-distribution is infeasible.

2.5 Digital Signatures

Digital signatures [Rab78] are a branch of public key cryptography. The notion was firstly described by Whitfield Diffie and Martin Hellman in 1976 [DH76]. Unlike the encryption schemes concentrating on the message’s confidentiality, digital signatures focus on the message’s integrity and the sender’s authenticity. In comparison to the handwritten signature, a digital signature is a binary string generated by applying the signing algorithm on a message associated with a private key. A typical digital signature scheme must satisfy two requirements: (1) A valid signature can only be generated by using the private key; (2) Only a valid message-signature pair can pass the verification under the corresponding public key.
Definition 2.13 A digital signature scheme includes three algorithms: \textit{KeyGen}, \textit{Sign}, and \textit{Verify}. \textit{KeyGen} is the key pair generation algorithm which takes a system random parameter $1^k$ as input and outputs a key pair,

$$(pk, sk) \leftarrow \textit{KeyGen}(1^k),$$

where $pk, sk$ are the public key and the private key respectively. \textit{Sign} is the signing algorithm run by the signer which takes a message $m$ and a private key $sk$ as input and outputs a signature $\sigma$,

$$\sigma \leftarrow \textit{Sign}(m, sk).$$

\textit{Verify} is the signature verifying algorithm which takes a message $m$, a signature $\sigma$ and a public key $pk$ as input and outputs true or false,

$$\textit{Verify}(m, \sigma, pk) \begin{cases} 
\text{true, if } \sigma \leftarrow \textit{Sign}(m, sk) \text{ and } (pk, sk) \leftarrow \textit{KeyGen}(1^k). \\
\text{false, otherwise.}
\end{cases}$$

The basic security requirement of a digital signature scheme is unforgeability. In other words, given a message, it must be infeasible to compute a valid signature without the knowledge of the private key. There are four different levels of breaking a digital signature scheme:

- **Total break:** the adversary is able to compute the private key.

- **Universal forgery:** the adversary can compute valid signatures of any message.

- **Selective forgery:** the adversary is able to compute a valid signature of a message chosen by himself.

- **Existential forgery:** the adversary is able to compute at least one valid message-signature pair which has not appeared before.

It is often required that a digital signature scheme is existential unforgeable against an adaptive chosen message attack, which is the strongest security notion.

## 2.6 Proxy Signatures

A proxy signature is a kind of digital signature where one party, denoted as the origin, delegates his signing power to another party, denoted as the proxy; therefore,
the proxy is able to sign messages on behalf of the origin. The notion of a proxy signature was introduced by Mambo, Usuda and Okamoto in 1996 [MUO96].

Based on the delegation type, the proxy signature can be divided into three categories: full delegation, partial delegation, and delegation by warrant. The full delegation means that the origin gives his private key to the proxy directly therefore the proxy has the same signing power as the origin. Obviously, this is insecure and impractical. The partial delegation means that the proxy possesses a proxy private key, which is different from the origin’s private key, and he can sign the messages on behalf of the origin using this proxy private key. The disadvantage is the proxy has the capability of signing any messages without the limitation. These weaknesses can be conquered in the delegation by warrant, where a delegation is generated by the origin to specify what kind of messages the proxy can sign.

Proxy signature schemes can be classified into proxy-unprotected and proxy-protected. In a proxy-protected scheme only the proxy can generate a valid proxy signature, while in a proxy-unprotected scheme either the origin or the proxy has the capability of generating a valid proxy signature. In practice, proxy-protected schemes are often required to avoid potential disputes between the origin and the proxy.

**Definition 2.14** A proxy signature scheme includes four algorithms: \texttt{KeyGen}, \texttt{DeleGen}, \texttt{ProxySign}, and \texttt{ProxyVer}. \texttt{KeyGen} is the key pair generation algorithm run by the origin or the proxy for generating their key pairs respectively,

\[
(pk, sk) \leftarrow \texttt{KeyGen}(1^k),
\]

where \( pk, sk \) are the public key and the private key respectively. \texttt{DeleGen} is the delegation generation algorithm run by the origin which takes a warrant \( w \), a private key \( sk_1 \) of the origin as input and outputs a delegation \( d \),

\[
d \leftarrow \texttt{DeleGen}(w, sk_1).
\]

\texttt{ProxySign} is the proxy signing algorithm run by the proxy which takes a message \( m \), a warrant \( w \), a delegation \( d \) of the warrant \( w \), and a private key \( sk_2 \) of the proxy as input and outputs a proxy signature \( \sigma \),

\[
\sigma \leftarrow \texttt{ProxySign}(m, w, d, sk_2).
\]

\texttt{ProxyVer} is the proxy signature verifying algorithm which takes a message \( m \), a warrant \( w \), a proxy signature \( \sigma \), a public key \( pk_1 \) of the origin, and a public key \( pk_2 \).
of the proxy as input and outputs true or false,

\[
ProxyVer(m, w, \sigma, pk_1, pk_2) \begin{cases} 
\text{true, if } \sigma \leftarrow ProxySign(m, w, d, sk_2), \\
\text{and } d \leftarrow DeleGen(w, sk_1).
\end{cases}
\]

\[
\text{false, otherwise.}
\]

2.7 Accumulators

Introduced by Benaloh and de Mare [BM94], accumulators allow a set of values being aggregated into a single value, where the size of the single value is independent of the size of the set. For each element in that set, there is an efficient computable witness value that shows this element has been aggregated into the accumulator value. The notion of the accumulator was extended to the dynamic accumulator by Camenisch and Lysyanskaya [CL02], where it requires efficiency to add an element into, or remove an element from, the accumulator value. The corresponding witness value is also required to be updated efficiently.

We adapt Nguyen’s [Ngu05] definition of the accumulator as follows.

**Definition 2.15** An accumulator is a scheme satisfying the following properties:

- **Efficient Generation.** There exists an efficient (in polynomial time) algorithm \text{Gen} which takes a security parameter \(1^k\) as input and outputs a tuple \((f, g, Y, u, t_f)\), where \(f : X_f \times Y^* \to X_f\) is a function and \(g : X_f \to X_g\) is a function for some domains \(X_f, X_g, Y^*\); \(u \in X_f\) is the initial value for accumulator; \(Y \subseteq Y^*\) is the domain for the elements which will be accumulated; \(t_f\) is some optional auxiliary information about \(f\); \(f, g\) are chosen randomly and uniformly from their domains respectively,

\[
(f, g, Y, u, t_f) \leftarrow \text{Gen}(1^k).
\]

- **Quasi-commutativity.** For all \(u \in X_f\), and \(y_1, y_2 \in Y^*\), we have \(f(f(u, y_1), y_2) = f(f(u, y_2), y_1)\). Therefore, for any \(Y = \{y_1, ..., y_l\} \subseteq Y^*\), we denote \(f(\ldots f(f(u, y_1), y_2), \ldots, y_l)\) as \(f(u, Y)\). We call \(g(f(u, Y))\) the accumulator value of the set \(Y\) over \(u\).

- **Efficient Evaluation.** For all \((f, g, Y, u, t_f) \leftarrow \text{Gen}(1^k), u \in X_f, \text{ and } Y \subseteq Y^*\) with polynomial bounded size, \(g(f(u, Y))\) is computable in polynomial time in \(k\).
We adapt the definition of dynamic accumulator presented in [ATSM09] as follows.

**Definition 2.16** A dynamic accumulator is an accumulator together with the following properties:

- **Accumulator Efficient Update.** For any accumulator value \( v = g(f(u, Y)) \) there exists an polynomial time algorithm \( B_1 \) such that: if there is a \( y \notin Y \), then we have \( v' \leftarrow B_1(t_f, v, y) \) such that \( v' = g(f(u, Y \cup \{y\})) \); if \( y \in Y \), we have \( v' = g(f(u, Y \setminus \{y\})) \).

- **Witnesses Efficient Update.** For any accumulator value \( v = g(f(u, Y)) \), any \( y \in Y \) associated with its witness \( w \) satisfying \( v = g(f(g^{-1}(w), y)) \), then there exists an polynomial time algorithm \( B_2 \) such that: if there is a \( y' \notin Y \) and \( v' \) is the updated accumulator with respect to \( y' \) be the newly added element, then we have \( w' \leftarrow B_2(v, v', y, y', w) \), where \( v' = g(f(g^{-1}(w'), y)) \); if there is a \( y' \in Y \) and \( v' \) is the updated accumulator with respect to \( y' \) be the newly removed element, then we have \( w' \leftarrow B_2(v, v', y, y', w) \), where \( v' = g(f(g^{-1}(w'), y)) \).

The security requirement of a dynamic accumulator scheme is that it is infeasible to find a witness with respect to an element which has not been accumulated into the accumulator. We call this property collision resistance.

### 2.8 Secret Handshakes

The secret handshake, introduced by Balfanz et al. [BDS+03], is an authentication procedure between two members of the same group for identifying each other secretly. The procedure will be successful if and only if both of two participants belong to the same group. In other words, the identity of each party will be revealed to the other party if and only if both parties are in the same group. An invalid party (group member) learns nothing about the other party’s affiliation.

**Definition 2.17** A secret handshake scheme includes three algorithms: CreateGroup, AddUser, and Handshake. The algorithm CreateGroup is run by a group manager which takes a random system parameter \( 1^k \) as input and outputs a key pair \((pk_G, sk_G)\) where \( pk_G \) is the group public key and \( sk_G \) is the group private key,

\[ (pk_G, sk_G) \leftarrow \text{CreateGroup}(1^k). \]
The algorithm \texttt{AddUser} is run by the group manager for adding a user into the group. It takes $pk_G, sk_G$, and the identity $ID$ of a user as input, and outputs a credential $\text{Cred}_{ID}$ showing the user's group membership,

$$\text{Cred}_{ID} \leftarrow \text{AddUser}(pk_G, sk_G, ID).$$

The authentication algorithm \texttt{HandShake} is executed between two parties. Take the public input which is their identities $ID_A, ID_B$ respectively, and the private input which is their credentials $\text{Cred}_{ID_A}, \text{Cred}_{ID_B}$ respectively, the algorithm outputs either accept or reject for each party,

$$\text{HandShake}(ID_A, ID_B, \text{Cred}_{ID_A}, \text{Cred}_{ID_B}) \begin{cases} \text{accept, if } ID_A, ID_B \text{ in a same group.} \\ \text{reject, otherwise.} \end{cases}$$

Generally, a secret handshake scheme is required to possess the following properties:

- \textit{Completeness}: The scheme always outputs \texttt{accept} if it is executed by the honest group members from a same group.

- \textit{Impersonator Resistance}: The impersonator resistance property is violated if an honest group member authenticates an adversary as a group member where the adversary is not a member of the group.

- \textit{Detector Resistance}: The detector resistance property is violated if an adversary can determine if a party is a member of a certain group through a handshake with this party.

Many secret handshakes schemes have been proposed so far. Readers may refer to [CJT04, ZSM06, ZSM07] for more information.

## 2.9 Oblivious Signature-Based Envelope

Oblivious Signature-Based Envelope (OSBE), introduced by Li \textit{et al.} [LDB03b], was proposed to conquer the policy cycles in automated trust negotiation (ATN). Informally, OSBE is a protocol which enables a sender to send an encrypted message to a receiver with the requirements: (1) the receiver can only open the encrypted message if he is granted some certification, which allows him to open such a message, from a third party; and (2) the sender can not learn any information about the receiver such as whether the receiver has the certification.
Definition 2.18 An Oblivious Signature-Based Envelope (OSBE) scheme consists of four parties: certificate authority (CA), sender $S$, receiver $R_1$, and receiver $R_2$. There are three phases of the communications between the parties:

- **Setup:** The CA runs a key generation algorithm which takes a security parameter $1^k$ as input and outputs a key pair $(pk, sk)$ where $pk$ is the public key and $sk$ is the corresponding private key. The CA picks a random message $x$ and runs the signing algorithm to generate a digital signature of $x$. Denote $\text{Sig}_{sk}(x)$ the signature of $x$ using the private key $sk$. The CA keeps the private key $sk$ secret. It gives the system parameters, the public key $pk$, and the message $x$ to $S$, $R_1$, and $R_2$. The CA gives $\text{Sig}_{sk}(x)$ to $R_1$.

- **Interaction:** The sender $S$ selects a message $M$ at first. After the sender $S$ interacts with the receivers $R_1$, $R_2$ respectively, both $R_1$ and $R_2$ obtain the identical encrypted message of $M$.

- **Open:** $R_1$ outputs the message $M$; $R_2$ does nothing.

Generally, an OSBE scheme is required to satisfy the following properties:

- **Soundness:** A receiver must be able to open the encrypted message in the open phase if he possesses the corresponding certificate (signature).

- **Obliviousness:** After the interaction phase finished, a sender can not tell whether a receiver he interacted with is able to open the encrypted message.

- **Semantically Secure:** A malicious receiver which does not possess an appropriate certificate (signature) learns nothing about the original message.

Numerous OSBE schemes have been proposed so far. They includes OSBE for RSA signatures [LDB03b], CA-OSBE [CJT04], OSBE for Schnorr signatures, and OSBE for Rabin signatures [NT05].

### 2.10 Proxy-Based Mobile Agents

The first notable implementation of secure mobile agents was proposed by Kotzanikolaou et al. [KBC00] in 2000. Their solution is based on the RSA cryptosystem. Their protocol includes three phases: setting, preparing an agent, and executing an agent.
1. Setting.

- The customer randomly selects two large primes \( p, q \), then computes a number \( n = p \times q \) and a number \( e \) \((1 < e < \phi(n) = (p - 1)(q - 1))\) such that \( \gcd(e, \phi(n)) = 1 \).
- Select a number \( d \) where \( 1 < d < \phi(n) = (p - 1)(q - 1) \) and \( de = 1 \mod \phi(n) \).
- Let \( H \) be an appropriate hash function (e.g. MD5).
- Let \( C \) denote the customer and \( req_C \) denote the warrant of the customer. Set \( h = H(C, req_C) \) a binary string whose value is bounded by \( n \). The warrant \( req_C \) defines the requirements of the customer for a specific purchase. Let \( S \) denote the server and \( bid_S \) denote the bid of the server, whose structure is analog to \( req_C \).

2. Preparing an Agent.

- The customer embeds the undetachable signature function pair

\[
f(.) = h^{(\cdot)} \mod n
\]

and

\[
f_{\text{signed}}(.) = k^{(\cdot)} \mod n
\]

into the mobile agent, where \( k = H^d \mod n \) is the customer’s RSA signature of \( h \). Here, \( f_{\text{signed}} \) is the encryption \( s \cdot f. \) \( s(.) = (\cdot)^d \mod n \) is the customer’s RSA signature function:

\[
f_{\text{signed}}(.) = s \cdot f(.) = s(f(.)) = s(h^{(\cdot)}) = (h^{(\cdot)})^d = (H^{d})^{(\cdot)} = k^{(\cdot)}.
\]

- The mobile agent then migrates to the server with \((f(.), f_{\text{signed}}(.))\) as part of its code and \((C, req_C)\) as part of its data.

3. Executing an Agent.

- With input \( x = H(S, C, bid_S) \), the server executes the mobile agent to obtain the RSA signature \((m, z)\):

\[
m = f(x) = h^x \mod n
\]

and

\[
z = f_{\text{signed}}(x) = k^x = (h^d)^x = (h^x)^d = m^d = s(m) \mod n.
\]
2.10. Proxy-Based Mobile Agents

- The mobile agent brings the transaction receipt \((S, C, \text{bid}_S, x, m, z)\) back to the customer.

The protocol described above is secure as the RSA cryptosystem. The main features of this protocol includes efficiency and transaction signature undetachable. A malicious server can not produce a valid transaction signature which breaks the requirements of the customer. However, the protocol is asymmetric. Only the customer signs the transaction using RSA whereas the server is not committed to the transaction. The repudiation issue is actually an issue in this protocol.

Proxy signatures are thought to be an appropriate solution for solving the repudiation issue. Therefore, a few mobile agent schemes based on proxy signatures were proposed. Lee et al. [LKK01b] proposed the first RSA-based and Schnorr-based secure mobile agent implementation by using a strong non-designated proxy signature.

Their RSA-based secure mobile agent implementation includes setting, preparing an agent, executing an agent, and verifying signature phases.

1. **Setting.** This phase is same as [KBC00]. Let \(A\) be a customer who has an authentic RSA key \((n_A, e_A, d_A)\) and \(B\) be a server who has authentic RSA key \((n_B, e_B, d_B)\). Let \(\text{ID}_A\) and \(\text{ID}_B\) denote the identities of \(A\) and \(B\), respectively. Let \(\text{req}_A\) be \(A\)'s requirement for a purchase and \(\text{bid}_B\) be \(B\)'s bid information which conforms to \(\text{req}_A\).

2. **Preparing an Agent.** \(A\) computes \(k = h(\text{ID}_A, \text{req}_A)^{d_A} \mod n_A\). \(A\) then embeds \((\text{ID}_A, \text{req}_A, k)\) into the mobile agent. The mobile agent will migrate to servers through the network.

3. **Executing an Agent.**
   - \(B\) verifies the validity of the mobile agent by checking 
     \[k^{e_A} \mod n_A \overset{?}{=} h(\text{ID}_A, \text{req}_A).\]
   - \(B\) generates a bid \(\text{bid}_B\) and computes 
     \[x = h(\text{ID}_A, \text{req}_A, \text{ID}_B, \text{bid}_B)^{d_B} \mod n_B.\]
   - \(B\) computes 
     \[y = h(\text{ID}_A, \text{req}_A)^x \mod n_A\] and 
     \[z = k^x \mod n_A.\]
   - \(B\) gives the following messages to the mobile agent as the transaction receipt.
     \[(\text{ID}_A, \text{req}_A, \text{ID}_B, \text{bid}_B, x, y, z).\]
4. Verifying the Signature (by anyone). When $A$ receives $(\text{ID}_A, req_A, \text{ID}_B, bid_B, x, y, z)$ from the mobile agent, he verifies the following:

- $B$’s signature: $x^e_B \mod n_B = h(\text{ID}_A, req_A, \text{ID}_B, bid_B)$.
- $y = h(\text{ID}_A, req_A)^x \mod n_A$.
- $A$’s signature: $z^e_A \mod n_A = y$.
- $\text{bid}_B \in \{req_A\}$.

The proxy signature is valid only when all the verifications above are passed.

Their Schnorr-based secure mobile agent implementation includes:

1. Setting. Let $A$ be a customer who has an authentic key pair $(x_A, y_A)$ and $B$ be a server who has authentic key pair $(x_B, y_B)$. The rest is same as in RSA-based protocol.

2. Preparing an Agent. $A$ selects a random number $k_A \in \mathbb{Z}_q^*$ and computes $r_A = g^{k_A}, s_A = x_Ah(req_A, r_A) + k_A$. $A$ gives $(req_A, r_A, s_A)$ to the mobile agent.

3. Executing an Agent. $B$ verifies the validity of the mobile agent by checking $g^{s_A} = y_A^{h(req_A, r_A)}r_A$. $B$ then generates a secure proxy key pair as

$$x_P = s_A + x_B, y_P = g^{x_P} = y_A^{h(req_A, r_A)}r_Ay_B.$$  

$B$ then signs $m = (\text{ID}_A, req_A, \text{ID}_B, bid_B, r_A)$ with the proxy private key $x_P$ to generate $\sigma_P = S(x_P, m)$ using the Schnorr signature scheme. After that, $B$ returns

$$(\text{ID}_A, req_A, \text{ID}_B, bid_B, r_A, \sigma_P)$$

to the mobile agent as the transaction receipt.

4. Verifying the Signature (by anyone). When $A$ receives $(\text{ID}_A, req_A, \text{ID}_B, bid_B, r_A, \sigma_P)$ from the mobile agent, he verifies the following:

- $V(y_P, m, \sigma_P) = \text{true}$, where $y_P = y_A^{h(req_A, r_A)}r_Ay_B$
  and $m = (\text{ID}_A, req_A, \text{ID}_B, bid_B, r_A)$.
- $\text{bid}_B \in \{req_A\}$.

The proxy signature is valid only when all the verifications above are passed.
These solutions indeed prevent the repudiation issue of the server in mobile agent applications. However, delegation misuse becomes a new issue, as any host in the network can generate a valid proxy signature by executing the mobile agent as long as the warrant is satisfied. Kim et al. [KBLK01] provided a scheme which limits the signing power of the remote host by applying one-time proxy signature. Nevertheless, their scheme cannot prevent the problem of delegation misuse. This actually leads to our motivation to develop a better solution.

2.11 Multi-Agents

Existing works focus only on how to secure the information exchanged among agents rather than the privacy of the negotiating agents themselves. Agent privacy protection in multi-agent systems have not drawn adequate attention and actually been ignored or mistreated. To the best of our knowledge, there is no such a appropriate solution. It is obviously that security and privacy issues among multi-agent systems are indeed significant and must be resolved properly. During our research, we found that these issues are actually hard to be solved in traditional multi-agents paradigm. Therefore, in order to solve these issues, a novel security approach was applied to construct our secure multi-agent scheme.
Chapter 3

Secure Mobile Agents with Designated Hosts

The inability of mobile agents to authenticate transactions in a hostile environment has always been considered as a major issue. It was believed unsolvable [ACCK01] until Kotzanikolaou et al. [KBC00] published a sound scheme. Their RSA-based secure mobile agent scheme is the first scheme which achieves authentication in a hostile environment. Proxy signatures, where the signing ability of an origin can be delegated to a proxy signer, can be applied to mobile agent security. In these schemes, the mobile agent can bring the delegation token of the home (original signer) to the remote hosts (proxy signer), thereby the remote hosts are able to generate a signed service under the delegation token. The problem is that any remote hosts can sign the service once they receive a delegation token. It actually exposes a problem that delegation could be abused by malicious hosts. In this chapter, we present a secure mobile agent scheme which addresses this problem.

In Section 3.1, we introduce our mobile agent architecture and the transaction procedure. It explains how our mobile agent works. We then provide our security model in Section 3.2. We defined three different types of adversaries in the security model: a) malicious hosts in the designated host list; b) malicious hosts not in the designated host list; and c) malicious customer. Each adversary was given a different level of attacking power, respectively. Then, we revisit two mathematical hard problems: Computational Diffie-Hellman Problem and \( q \)-Strong Diffie-Hellman Problem. The security of our scheme is based on the hardness of these two mathematical problems. In Section 3.3, we present our scheme, followed by the security proof of our scheme in Section 3.4. We summarise this chapter in Section 3.5.
3.1 Task Execution Procedure of Proposed Mobile Agent

In general, our mobile agent system consists of the following phases: Customer Setup, Agent Setup, Agent Dispatch, Host Execution, and Verification.

- **Customer Setup:** The customer decides the services he intends to receive and selects a set of hosts for inclusion.

- **Agent Setup:** The customer generates a delegation token based on its requirements and embeds it in the mobile agent. This token includes the list of designated hosts that are permitted in the agent network.

- **Agent Dispatch:** The mobile agent travels around the network and searches for host from the list.

- **Host Execution:** When a mobile agent arrives at a host, the host checks the validity of the delegation token. If it is invalid, the host will stop the execution; otherwise, it will execute the mobile agent following the designated host procedure.

- **Verification:** Anyone can verify whether the signed service is valid, following the verification algorithm.

Figure 3.1 demonstrates an example of a mobile agent executing a task on behalf of its owner in a general online application. The customer, as the initiator, selects a set of hosts (1, 2, and 5) for inclusion. The customer then generates a delegation token based on the task and the designated host list, and embeds the token in a mobile agent. The mobile agent then travels around the network searching for the designated hosts from the list. When an agent arrives at a host, say Host 1, Host 1 verifies the delegation token prior to an execution. Because Host 1 is in the list, it can be validated and offer a signed service agreement satisfying the task defined by the customer. Hosts 3, 4, 6, and 7 are excluded from the agent network, because they are not defined in the delegation token.
### 3.2 Security Model

We consider two types of adversary: (1) malicious hosts and (2) malicious customer. Type (1) is classified into two subtypes: (a) malicious hosts that are in the designated host list and (b) malicious hosts that are not in the designated host list. To prove security in our system, we will show that, for (a), malicious hosts are not able to generate a valid signed service under the delegation token which is unknown to them; and for (b), malicious hosts possess the delegation token, but they are not able to generate a valid signed service.

#### 3.2.1 Existential Unforgeability Against Malicious Hosts

In this case, we assume that the adversary can query the private key of any host it chooses, but cannot query the private key of the customer. We allow all the malicious hosts to collude, i.e., the adversary can obtain all private keys of the hosts in the agent network.

**Malicious hosts that are in the designated host list**

Here, we will show that, the probability of an adversary $A_1$ to output a valid signature under the delegation token that is unknown to him is negligible. It is defined
using the following game between a challenger $C$ and an adversary $A_1$:

- **Setup**: $C$ runs the ParaGen algorithm to obtain system’s parameters $\text{Param}$.

- **Key extract queries**: Given an identity $\text{ID}$ of a host chosen by $A_1$, $C$ returns the private key $s_{\text{ID}}$ corresponding to $\text{ID}$.

- **Delegation queries**: $A_1$ can choose the warrant $w$ adaptively and the designated host list $X = (H(\text{ID}_1), H(\text{ID}_2), ..., H(\text{ID}_k))$ adaptively, and submit these to $C$. Here, $k \leq \theta$ and $\theta$ is the upper bound defined in $\text{Param}$. In response, $C$ runs the algorithm $D \leftarrow \text{DeleGen}(w, X)$ and returns $D$ to $A_1$. $D$ is the delegation token resulting from the combination of $X$ and $w$ chosen by $A_1$.

- **Signature queries**: Proceeding adaptively, $A_1$ can request the signature of $(m, w, X, \text{ID}_i)$, where $m$ is the message (which is the offer in our case) that the host needs to sign, $w$ is the customer’s warrant, $X$ represents the designated host list, $\text{ID}_i$ is the identity of the host who signs the message, and $\text{ID}_i$ is in the designated host list. In response, $C$ runs the algorithm to get the delegation $D \leftarrow \text{DeleGen}(w, X)$, then runs the algorithm $\sigma \leftarrow \text{Sign}(s_{\text{ID}_i}, m, D)$, and returns $\sigma$ to $A_1$.

- **Output**: Finally, $A_1$ outputs $(m^*, w^*, X^*, \text{ID}^*, \sigma^*)$ and wins the game if:
  1. $\text{ID}^*$ is an identity of a host which is in the designated host list $X^*$.
  2. $(w^*, X^*)$ has not been requested as one of the Delegation queries.
  3. $(m^*, w^*, X^*, \text{ID}^*)$ has not been requested as one of the Signature queries.
  4. $\text{Verify}(\text{Param}, PK_C, PK_{\text{ID}^*}, m^*, w^*, X^*, \text{ID}^*, \sigma^*) = \text{valid}$.

We define $\text{Succ}_{A_1}$ to be the probability that the adversary $A_1$ wins the above game.

**Definition 3.1** We say an adversary $A_1$ can $(t, q_{H^*}, q_{KE}, q_D, q_S, \epsilon)$-break this scheme if $A_1$ runs in time at most $t$, $A_1$ makes at most $q_{H^*}$ hash queries, at most $q_{KE}$ key extract queries, at most $q_D$ delegation queries, and at most $q_S$ signature queries, $\text{Succ}_{A_1}$ is at least $\epsilon$. 
Malicious hosts that are not in the designated host list

In this situation, the goal of an adversary $A_2$ is to output a valid signature with the host that is not in the designated host list. We will show that, even $A_2$ possesses the delegation token, the probability of $A_2$ to output a valid signature under the delegation token is still negligible. It is defined using the following game between a challenger $C$ and an adversary $A_2$. After all the queries:

- **Output**: $A_2$ outputs $(m^*, w^*, X^*, ID^*, \sigma^*)$ and wins the game if:
  1. $ID^*$ is an identity of a host which is not in the designated host list $X^*$.
  2. $Verify(Param, PK_C, PK_{ID^*}, m^*, w^*, X^*, ID^*, \sigma^*) = valid$.

We define $Succ_{A_2}$ to be the probability that the adversary $A_2$ wins the above game.

**Definition 3.2** We say an adversary $A_2$ can $(t, q_{H^*}, q_{KE}, q_D, q_S, \epsilon)$-break this scheme if $A_2$ runs in time at most $t$, $A_2$ makes at most $q_{H^*}$ hash queries, at most $q_{KE}$ key extract queries, at most $q_D$ delegation queries, and at most $q_S$ signature queries, $Succ_{A_2}$ is at least $\epsilon$.

### 3.2.2 Existential Unforgeability Against Malicious Customer

We assume that a malicious customer possesses the private keys $sk_C$ and $t$. We want to show that a malicious customer cannot generate a valid signature for a host. Given a valid signature, the host cannot deny the fact that he has signed the message (transaction or service). It is defined using the following game between a challenger $C$ and an adversary $A_3$. After all the queries:

- **Output**: $A_3$ outputs $(m^*, w^*, X^*, ID^*, \sigma^*)$ and wins the game if:
  1. $ID^*$ is an identity of a host which is in the designated host list $X^*$.
  2. $(m^*, w^*, X^*, ID^*)$ has not been requested as one of the Signature queries.
  3. $Verify(Param, PK_C, PK_{ID^*}, m^*, w^*, X^*, ID^*, \sigma^*) = valid$.

We define $Succ_{A_3}$ to be the probability that the adversary $A_3$ wins the above game.

**Definition 3.3** We say an adversary $A_3$ can $(t, q_{H^*}, q_S, \epsilon)$-break this scheme if $A_3$ runs in time at most $t$, $A_3$ makes at most $q_{H^*}$ hash queries, and at most $q_S$ signature queries, $Succ_{A_3}$ is at least $\epsilon$. 
3.3. Our Scheme

3.2.3 Complexity Definitions

We recall the complexity definitions and the assumptions which will be used in our security proofs.

Definition 3.4 (Computational Diffie-Hellman (CDH) on $G_1$.) Given $g, g^a, g^b \in G_1$, for some unknown $a, b \overset{\$}{\leftarrow} \mathbb{Z}_p$, compute $g^{ab} \in G_1$.

Definition 3.5 (Computational Diffie-Hellman (CDH) Assumption on $G_1$.) Given $g, g^a, g^b \in G_1$, for some unknown $a, b \overset{\$}{\leftarrow} \mathbb{Z}_p$, the following function $\text{Succ}^{\text{CDH}}_{A,G_1}$ is negligible for any polynomially bounded algorithm $A$.

$$\text{Succ}^{\text{CDH}}_{A,G_1} = \Pr[A(g, g^a, g^b) = g^{ab} : a, b \overset{\$}{\leftarrow} \mathbb{Z}_p].$$

Definition 3.6 (q-Strong Diffie-Hellman (q-SDH) on $G_1$.) Given a tuple $(g, g^s, ..., g^{s_q})$, for a unknown $s \overset{\$}{\leftarrow} \mathbb{Z}_p^*$ and $q \in \mathbb{Z}_p^*$, compute a pair $(c, g^{\frac{1}{s+c}})$ where $c \in \mathbb{Z}_p$.

Definition 3.7 (q-Strong Diffie-Hellman (q-SDH) Assumption on $G_1$.) Given a tuple $(g, g^s, ..., g^{s_q})$, for a unknown $s \overset{\$}{\leftarrow} \mathbb{Z}_p^*$ and $q \in \mathbb{Z}_p^*$, the following function $\text{Succ}^{\text{q-SDH}}_{A,G_1}$ is negligible for any polynomially bounded algorithm $A$.

$$\text{Succ}^{\text{q-SDH}}_{A,G_1} = \Pr[A(g, g^s, ..., g^{s_q}) = ((c, g^{\frac{1}{s+c}}) \land c \in \mathbb{Z}_p)].$$

3.3 Our Scheme

We now construct our scheme using bilinear maps in the random oracle model.

1. Setup:

Select $(G_1, G_2)$ as cyclic multiplicative groups where $|G_1| = |G_2| = p$ for some prime $p$. Let $g$ be a generator of $G_1$. Define the bilinear map $\hat{e} : G_1 \times G_1 \rightarrow G_2$. Select four distinct secure hash functions: $H, H_0, H_1$ and $H_2$ where $H : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$, $H_0, H_1, H_2 : \{0,1\}^* \times G_1 \rightarrow G_1$. The system parameter is $\text{Param} = (G_1, G_2, p, g, \hat{e}, H, H_0, H_1, H_2)$.

2. Customer setup:

- Select a random number $sk_C \in \mathbb{Z}_p^*$ as his private key and compute the corresponding public key $pk_C = g^{sk_C} \in G_1$. 

• Randomly select $t \in \mathbb{Z}_p^*$ and compute a tuple $T = (g, g^t, g^{t^2}, g^{t^3}, \ldots, g^{t^\theta})$, where $\theta$ is the upper bound. e.g. the customer can only build a designated hosts list that includes $k$ hosts where $k \leq \theta$.

• Compute $E = g^{sk_C}$.

• Publish $PK_C = (E, T, pk_C)$ as the customer’s public key and keep $sk_C, t$ as his private keys.

3. Agent Setup:

• Build the designated host list:
  
  – The customer chooses $k$ hosts, $\text{ID}_1, \text{ID}_2, \ldots, \text{ID}_k$, and computes $X = (H(\text{ID}_1), H(\text{ID}_2), \ldots, H(\text{ID}_k)) \subset \mathbb{Z}_p \setminus \{-t\}$ where $\text{ID}_i$ is the identity of a host and $k \leq \theta$.
  
  – Compute
    \[ V = g^{\prod_{i=1}^k (H(\text{ID}_i) + t)} \in \mathbb{G}_1. \]

• The customer computes the delegation token $D = (H_0(w, V), V)^{sk_C}$ where $w \in \{0, 1\}^*$ is his warrant, $V$ is a value representing the designated host list.

• The customer embeds $(w, X, D)$ in the mobile agent.

4. Host Execution:

• Select a random number $s_{\text{ID}_i} \in \mathbb{Z}_p^*$ as his private key where $\text{ID}_i$ is his identity. The public key of this host is $P_{\text{ID}_i} = g^{s_{\text{ID}_i}} \in \mathbb{G}_1$.

• Verify the delegation token by checking
  \[ \hat{e}(D, g) \overset{?}{=} \hat{e}(H_0(w, V), pk_C)\hat{e}(V, pk_C), \]
  where
  \[ V = g^{\prod_{i=1}^k (H(\text{ID}_i) + t)} \]
  can be computed using $T$ and $X$ without $t$.

• If invalid, the host stops the execution. Otherwise,

  • the host computes $H(\text{ID}_i)$,
    
    – if $H(\text{ID}_i) \notin X$, the host stops the execution. Otherwise,
the host randomly chooses \( r \in \mathbb{Z}_p^* \) and generates the signature \( \sigma = (\Sigma, W_{ID}, R) \) where

\[
\begin{align*}
\Sigma &= D \cdot H_1(m||w||ID, V)^{sk_1} \cdot H_2(m||w||ID, V)^r, \\
W_{ID} &= g^{\prod_{h \in X}^{1{(h+t)}}}, \\
R &= g^r.
\end{align*}
\]

Here,

\[
W_{ID} = g^{\prod_{h \in X}^{1{(h+t)}}}
\]

can be computed using \( T \) and \( X \) without \( t \). \( m \in \{0, 1\}^* \) is the message (the offer of the host for the transaction) needs to be signed.

5. **Verification:** Given \( Param \), public keys \( PK_C, P_{ID} \), a host’s identity \( ID \), a warrant \( w \), a designated host list \( X \), a message \( m \), and a signature \( \sigma = (\Sigma, W_{ID}, R) \), verify that

\[
\hat{e}(\Sigma, g) = \hat{e}(H_0(w, V), pk_C) \cdot \hat{e}(W_{ID}, p_{ID}^{H(ID)} \cdot E) \cdot \\
\hat{e}(H_1(m||w||ID, V), P_{ID}) \cdot \hat{e}(H_2(m||w||ID, V), R).
\]

Here,

\[
V = g^{\prod_{h \in X}^{1{(h+t)}}}
\]

can be computed using \( X \) and \( PK_C \) without \( t \). If the equation holds, \( \sigma \) will be accepted as a valid signed service; otherwise, rejected.

The correctness of the scheme can be verified:

\[
\hat{e}(\Sigma, g) = \hat{e}(D, g) \cdot \hat{e}(H_1(m||w||ID, V)^{sk_1}, g) \cdot \hat{e}(H_2(m||w||ID, V)^r, g) \\
= \hat{e}((H_0(w, V) \cdot V)^{sk_C}, g) \cdot \hat{e}(H_1(m||w||ID, V), P_{ID}) \cdot \\
\hat{e}(H_2(m||w||ID, V), R) \\
= \hat{e}(H_0(w, V), pk_C) \cdot \hat{e}(W_{ID}, p_{ID}^{H(ID)} \cdot E) \cdot \\
\hat{e}(H_1(m||w||ID, V), P_{ID}) \cdot \hat{e}(H_2(m||w||ID, V), R).
\]

### 3.4 Security Analysis

#### 3.4.1 Existential Unforgeable Against Adversary \( A_1 \)

**Theorem 3.1** If there exists an adversary \( A_1 \) who can \( (t, q_{H^*}, q_{KE}, q_{ID}, q_S, \epsilon) \)-break the proposed scheme then there exists another algorithm \( B \) who can use \( A_1 \) to solve
an instance of the CDH problem in \( G_1 \) with probability

\[
\text{Succ}_{\mathcal{B}, G_1} \geq (1 - \frac{2}{q_D + q_S + 2})^{q_D + q_S}(\frac{2}{q_D + q_S + 2})^2 \epsilon
\]

in time \( t + c_{(G_1, G_2)}(q_H + q_D + q_S + q_{KE} + 1) \). Here \( c_{(G_1, G_2)} \) is a constant that depends on \((G_1, G_2)\).

**Proof.** Algorithm \( \mathcal{B} \) is given a random instance \((g, g^a, g^b)\) of the CDH problem in \( G_1 \). Its goal is to compute \( g^{ab} \) by interacting with adversary \( \mathcal{A}_1 \). \( \mathcal{B} \) will simulate the challenger and interact with \( \mathcal{A}_1 \) as described below. The hash functions \( H, H_0, H_1, H_2 \) are regarded as the random oracles during the proof.

**Setup:**

- Run the \texttt{ParaGen} algorithm to obtain the system’s parameters \texttt{Param}.
- Set \( sk_C = a \), therefore \( pk_C = g^a \).
- Maintain five lists: \texttt{H-list}, \texttt{H0-list}, \texttt{H1-list}, \texttt{H2-list} and \texttt{Key-list} that store the results of queries to random oracles respectively. Initially, they are empty.
- Randomly select \( t \in \mathbb{Z}_p^* \) and compute a tuple \( T = (g, g^t, g^{t^2}, g^{t^3}, \ldots, g^{t^\theta}) \) where \( \theta \in \mathbb{Z}_p^* \) is the upper bound of designated host list. Compute \( E = g^{t \cdot sk_C} = (g^a)^t \).
- Return \texttt{Param} and \((E, T, pk_C)\) to \( \mathcal{A}_1 \).

**H queries:** Adversary \( \mathcal{A}_1 \) can make queries to the \texttt{H} oracle of the input \( \mathsf{ID}_i \) at any time, \( \mathcal{B} \) checks the \texttt{H-list} first:

- If there exists an item \((\mathsf{ID}_i, h_i)\) in the list, \( \mathcal{B} \) will return \( h_i \) to \( \mathcal{A}_1 \).
- Otherwise, \( \mathcal{B} \) randomly chooses \( h_i \in \mathbb{Z}_p^* \) such that there is no \((\cdot, h_i)\) entry in the list.

Then, \( \mathcal{B} \) returns \( h_i \) to \( \mathcal{A}_1 \) and adds \((\mathsf{ID}_i, h_i)\) to the \texttt{H-list}.

**H0 queries:** Proceeding adaptively, adversary \( \mathcal{A}_1 \) can make queries to the \texttt{H0} oracle of the input \((w_i, V_j)\) where \( w_i \in \{0, 1\}^* \) and \( V_j \in G_1 \). For a query \((w_i, V_j)\), \( \mathcal{B} \) checks the \texttt{H0-list} as follows:

- If there exists an item \(((w_i, V_j), h_{0ij}, c_{ij}, coin_{0ij})\) in the list, \( \mathcal{B} \) will return \( h_{0ij} \) to \( \mathcal{A}_1 \).
3.4. Security Analysis

Otherwise, $B$ tosses a coin $coin_{0ij} \in \{0, 1\}$ such that $Pr[coin_{0ij} = 1] = \delta$ (the value of $\delta$ will be determined later).

- If $coin_{0ij} = 1$, $B$ chooses $c_{ij} \leftarrow \mathbb{Z}_p^*$ and computes $h_{0ij} = g^b \cdot g^{c_{ij}}$.
- Otherwise, $coin_{0ij} = 0$, $B$ chooses $c_{ij} \leftarrow \mathbb{Z}_p^*$ and computes $h_{0ij} = g^{c_{ij}}$.

If $h_{0ij}$ has already been in the tuple $(., h_{0ij}, ..)$ of the $H_0$-list, then $B$ chooses another $c_{ij}$ and recomputes $h_{0ij}$. $B$ then returns $h_{0ij}$ to $A_1$ and adds $((w_i, V_j), h_{0ij}, c_{ij}, coin_{0ij})$ to the $H_0$-list.

**H1 queries:** Proceeding adaptively, adversary $A_1$ can make queries to the $H_1$ oracle of the input $(m_i, w_j, \|D_k, V_i\|)$. $B$ maintains an $H_1$-list which consists of tuples $(\Omega_{ijkl}, h_{1ijkl})$. For a query $\Omega_{ijkl} = (m_i || w_j || D_k, V_i)$, $B$ checks the $H_1$-list as follows:

- If there exists an item $(\Omega_{ijkl}, h_{1ijkl})$ in the list, $B$ will return $h_{1ijkl}$ to $A_1$.
- Otherwise, $B$ randomly chooses $h_{1ijkl} \in \mathbb{G}_1$ such that there is no $(., h_{1ijkl})$ entry in the list.

Then, $B$ returns $h_{1ijkl}$ to $A_1$ and adds $(\Omega_{ijkl}, h_{1ijkl})$ to the $H_1$-list.

**H2 queries:** Proceeding adaptively, adversary $A_1$ can make queries to the $H_2$ oracle of the input $(m_i, w_j, \|D_k, V_i\|)$. $B$ maintains an $H_2$-list which consists of tuples $(\Omega_{ijkl}, h_{2ijkl}, e_{ijkl}, coin_{2ijkl})$. For a query $\Omega_{ijkl} = (m_i || w_j || D_k, V_i)$, $B$ checks $H_2$-list as follows:

- If there exists an item $(\Omega_{ijkl}, h_{2ijkl}, e_{ijkl}, coin_{2ijkl})$ in the list, $B$ will return $h_{2ijkl}$ to $A_1$.
- Otherwise, $B$ tosses a coin $coin_{2ijkl} \in \{0, 1\}$ such that $Pr[coin_{2ijkl} = 1] = \delta$.
  - If $coin_{2ijkl} = 1$, $B$ chooses $e_{ijkl} \leftarrow \mathbb{Z}_p^*$ and computes $h_{2ijkl} = g^{e_{ijkl}}$.
  - Otherwise, $coin_{2ijkl} = 0$, $B$ chooses $e_{ijkl} \leftarrow \mathbb{Z}_p^*$ and computes $h_{2ijkl} = g^{e_{ijkl}} / g^b$.

If $h_{2ijkl}$ has already been in the tuple $(., h_{2ijkl}, ..)$ of the $H_2$-list, then $B$ chooses another $e_{ijkl}$ and recomputes $h_{2ijkl}$. After that, $B$ returns $h_{2ijkl}$ to $A_1$ and adds $(\Omega_{ijkl}, h_{2ijkl}, e_{ijkl}, coin_{2ijkl})$ to the $H_2$-list.

**Key extract queries:** In this process, $A_1$ can ask the private key of any host. For a query whose identity is $\|D_i, B$ checks the Key-list:
• If there is an item \((\text{ID}_i, s_{\text{ID}_i})\) in the Key-list, \(B\) will return \(s_{\text{ID}_i}\) to \(A_1\).

• Otherwise, \(B\) chooses \(s_{\text{ID}_i} \leftarrow \mathbb{Z}_p^*\).

If \(s_{\text{ID}_i}\) has already been in the tuple \((., s_{\text{ID}_i})\) of the Key-list, then \(B\) chooses another \(s_{\text{ID}_i}\). After that, \(B\) returns \(s_{\text{ID}_i}\) to \(A_1\) and adds \((\text{ID}_i, s_{\text{ID}_i})\) to the Key-list.

**Delegation queries:** Proceeding adaptively, \(A_1\) can ask at most \(q_D\) delegation queries of tuple \((w_i, X_j)\) chosen by itself. For such a query \((w_i, X_j)\), \(B\) first computes accumulator value

\[
V_j = g^{\prod_{h \in X_j} (h+\ell)}.
\]

We assume that there is a tuple \(((w_i, V_j), h_{0ij}, c_{ij}, \text{coin}_{0ij})\) in \(H_0\)-list containing \((w_i, V_j)\). \(B\) can make an \(H_0\) query \((w_i, V_j)\) if that tuple does not exist.

• If \(\text{coin}_{0ij} = 0\), then \(H_0(w_i, V_j) = h_{0ij} = g^{c_{ij}}\). \(B\) can compute

\[
D = (H_0(w_i, V_j) \cdot V_j)^a = (g^{c_{ij}})^a \cdot V_j^a = pk_{C_{ij}}^{c_{ij}} \cdot (g^{\prod_{h \in X_j} (h+\ell)})^a
\]

\[
= pk_{C_{ij}}^{c_{ij}} \cdot pk_{C}^{\prod_{h \in X_j} (h+\ell)}.
\]

• If \(\text{coin}_{0ij} = 1\), \(B\) terminates the simulation and reports failure.

**Signature queries:** In this process, \(A_1\) can ask at most \(q_S\) signature queries of his choice. For a query \((m_i, w_j, \text{ID}_k, X_i)\) where \(\text{ID}_k\) is in the designated host list \(X_i\). \(B\) first computes accumulator value \(V_i = g^{\prod_{h \in X_i} (h+\ell)}\). We assume there exists tuples \(((w_j, V_i), h_{0ij}, c_{ij}, \text{coin}_{0ij}), (\Omega_{ijkl}, h_{ijkl}), (\Omega_{ijkl}, h_{ijkl}, c_{ijkl}, \text{coin}_{2ijkl}), (\text{ID}_k, s_{\text{ID}_k}), (\text{ID}_k, h_k)\) in the \(H_0\)-list, \(H_1\)-list, \(H_2\)-list, Key-list and \(H\)-list respectively. Otherwise, \(B\) can make those queries by itself. Here, \(\Omega_{ijkl} = (m_i || w_j || \text{ID}_k, V_i)\).

• If \(\text{coin}_{0ij} = 0\), then \(H_0(w_j, V_i) = h_{0ij} = g^{c_{ij}}\). \(B\) can compute

\[
D = (H_0(w_j, V_i) \cdot V_i)^a = (g^{c_{ij}})^a \cdot V_i^a = pk_{C_{ij}}^{c_{ij}} \cdot (g^{\prod_{h \in X_i} (h+\ell)})^a
\]

\[
= pk_{C_{ij}}^{c_{ij}} \cdot pk_{C}^{\prod_{h \in X_i} (h+\ell)}.
\]

After that, \(B\) can generate a valid signature as described in **Host execution** in our scheme.

• If \(\text{coin}_{0ij} = 1\) and \(\text{coin}_{2ijkl} = 0\), then \(H_0(w_j, V_i) = g^{b} g^{c_{ij}}\) and \(H_2(m_i || w_j || \text{ID}_k, V_i) = g^{c_{ijkl}} / g^{b}\). \(B\) can generate a valid signature by setting \(R = g^{r} = g^{u} \cdot g^{a}\) where
3.4. Security Analysis

\[ u \xleftarrow{\$} \mathbb{Z}_p^*, \text{ therefore} \]

\[ \Sigma_{ijkl} = \left( H_0(w_i, V_i) \cdot V_i \right)^a \cdot H_1(m_i||w_j||ID_i, V_i)^{s_{idk}} \cdot H_2(m_i||w_j||ID_i, V_i)^r \]

\[ = \left( g^b \cdot g^{c_{ijl}} \right)^a \cdot V_i^a \cdot h_{ijkl}^{s_{idk}} \cdot (g^{e_{ijkl}} / g^b)^r \]

\[ = g^{ab} \cdot pk_{C}^{c_{ijl}} \cdot V_i^a \cdot h_{ijkl}^{s_{idk}} \cdot g^{(e_{ijkl} - b)(u + a)} \]

\[ = g^{ab} \cdot pk_{C}^{c_{ijl}} \cdot V_i^a \cdot h_{ijkl}^{s_{idk}} \cdot (g^{ue_{ijkl}} / g^{ab}) \cdot (g^{ae_{ijkl}} / g^{ab}) \]

\[ = pk_{C}^{c_{ijl}} \cdot pk_{C}^{\pi_{h \in X_1^{(h+t)}}} \cdot h_{ijkl}^{s_{idk}} \cdot (g^{ue_{ijkl}} / g^{ab}) \cdot pk_{C}^{e_{ijkl}} \]

\[ W_{idk} = g^{\Pi_{h \in X_1^{(h+t)}}}, \]

and \( R = g^r \). \( \mathcal{B} \) then returns \((\Sigma_{ijkl}, W_{idk}, R)\) which is a valid signature to \( \mathcal{A}_1 \).

- if \( coin_{0ijl} = 1 \) and \( coin_{2ijkl} = 1 \), \( \mathcal{B} \) terminates the simulation and reports failure.

If \( \mathcal{B} \) does not abort during all the queries, \( \mathcal{A}_1 \) will output a valid message-signature tuple \((m^*, w^*, X^*, ID^*, \sigma^*)\) with successful probability at least \( \epsilon \). Here, \( \sigma^* = (\Sigma^*, W^*, R^*) \) is a valid signature, \( ID^* \) is the identity of the host, \( w^* \) is the warrant, \( X^* \) is the designated host list and \( m^* \) is the message, such that \( ID^* \) is in the designated host list \( X^* \), \( (w^*, X^*) \) has not been requested as one of the delegation queries, \((m^*, w^*, X^*, ID^*)\) has not been requested as one of the signature queries. We assume there exists tuples \((ID^*, s_{ID^*}), (ID^*, h^*), ((w^*, V^*), h_{0s}, e^*, coin_{0s}), ((m^*||w^*||ID^*, V^*), h_{1s}) \) and \((m^*||w^*||ID^*, V^*), h_{2s}, e^*, coin_{2s})\) in the Key-list, H-list, H0-list, H1-list and H2-list respectively. Otherwise, \( \mathcal{B} \) can make those queries by itself.

- If \( coin_{0s} = 0 \) or \( coin_{2s} = 0 \), \( \mathcal{B} \) terminates the simulation and reports failure.

- Otherwise, \( coin_{0s} = 1 \) and \( coin_{2s} = 1 \). For this case, \( H_0(w^*, V^*) = h_{0s} = g^b \cdot g^{c^*}, H_2(m^*||w^*||ID^*, V^*) = h_{2s} = g^{e^*}, \sigma^* = (\Sigma^*, W^*, R^*) \) is a valid signature. Therefore,

\[ \Sigma^* = (g^b \cdot g^{c^*} \cdot V^*)^a \cdot h_{1s}^{s_{id}} \cdot g^{e^*} \cdot e^* \]

\[ = g^{ab} \cdot pk_{C}^{c_{ijl}} \cdot (V^*)^a \cdot h_{1s}^{s_{id}} \cdot (R^*)^{e^*} \]

and \( g^{ab} = \Sigma^*/(pk_{C}^{c_{ijl}} \cdot pk_{C}^{\pi_{h \in X^{(h+t)}}} \cdot h_{1s}^{s_{id}} \cdot (R^*)^{e^*}) \). Therefore, \( \mathcal{B} \) successfully solves the given instance of the CDH problem in \( G_1 \).

Now, we will show the successful probability for \( \mathcal{B} \). \( \mathcal{B} \) can output \( g^{ab} \) successfully if and only if:
3.4. Security Analysis

- \( B \) does not abort during the **Delegation queries.** This probability is \((1 - \delta)^q_D;\)
- \( B \) does not abort during the **Signature queries.** This probability is \((1 - \delta^2)^q_S;\)
- \( A_1 \) outputs a valid signature. This probability is greater than \( \epsilon; \)
- \( \text{coin}_{0*} = 1 \) and \( \text{coin}_{2*} = 1 \), this probability is \( \delta^2. \)

The probability of \( B \) can successfully output \( abP \) is

\[
\text{Succ}^{\text{CDH}}_{B,G_1} \geq (1 - \delta)^q_D (1 - \delta^2)^q_S \delta^2 \epsilon \\
\geq (1 - \delta)^q_D + q_S \delta^2 \epsilon,
\]

where \( \delta = \frac{2}{q_D + q_S + 2}. \) It is maximized as

\[
\text{Succ}^{\text{CDH}}_{B,G_1} \geq (1 - \frac{2}{q_D + q_S + 2})^{q_D + q_S} (\frac{2}{q_D + q_S + 2})^2 \epsilon.
\]

Algorithm \( B \)'s running time is the same as \( A \)'s running time plus the time it takes to respond to \( q_H \) random oracle queries, \( q_D \) delegation queries, \( q_S \) signature queries, \( q_{KE} \) key extraction queries and compute \( g^{ab} \) from \( \sigma^* \). We assume that each query requires at most time \( c_{(G_1,G_2)} \) which is a constant depending on the bilinear group pair \((G_1,G_2)\). Hence, the total running time is at most \( t + c_{(G_1,G_2)}(q_H^* + q_D + q_S + q_{KE} + 1) \).

This completes the proof.

### 3.4.2 Existential Unforgeable Against Adversary \( A_2 \)

**Theorem 3.2** If there exists an adversary \( A_2 \) who can \((t,q_H^*,q_{KE},q_D,q_S,\epsilon)-\)break the proposed scheme, then there exists another algorithm \( B \) that can use \( A_2 \) to solve an instance of the \( q \)-SDH problem in \( G_1 \) with the probability

\[
\text{Succ}^{q\text{-SDH}}_{B,G_1} = \text{Succ}_{A_2} \geq \epsilon
\]

in time \( t + c_{(G_1,G_2)}(q_H^* + q_D + q_S + q_{KE} + 1) \). Here \( c_{(G_1,G_2)} \) is a constant that depends on \((G_1,G_2)\).

**Proof.** Algorithm \( B \) is given a random instance \((g, g^x, g^{x^2}, ..., g^{x^q})\) of the \( q \)-SDH problem in \( G_1 \). Its goal is to compute \((c, g^{x^2})\) by interacting with adversary \( A_2 \). \( B \) will simulate the challenger and interact with \( A_2 \) as described below. The hash functions \( H, H_0, H_1, H_2 \) are regarded as the random oracles during the proof.

**Setup:**
3.4. Security Analysis

- Run the ParaGen algorithm to obtain the system’s parameters $Param$.
- Randomly choose $sk_C \in \mathbb{Z}_p^*$ as the customer’s private key, therefore $pk_C = g^{sk_C}$.
- Maintain five lists: $H$-list, $H_0$-list, $H_1$-list, $H_2$-list and Key-list, as in the proof of Theorem 3.1.
- Set $T = (g, g^x, g^{x^2}, ..., g^{x^q})$ where $q = \theta$ which is the upper bound of the designated host list, and compute $E = (g^x)^{sk_C}$.
- Return $Param$ and $(E, T, pk_C)$ to $A_2$.

**H queries:** As in the proof of Theorem 3.1, $A_2$ requests $\text{ID}_i$, $B$ returns $h_i$ and adds ($\text{ID}_i, h_i$) to the $H$-list if there is no such entry in the list.

**H0 queries:** As in the proof of Theorem 3.1, $A_2$ requests $(w_i, V_j)$, $B$ returns $h_{0ij} \leftarrow \mathcal{G}_1$ to $A_2$ and adds $((w_i, V_j), h_{0ij})$ to the $H_0$-list if there is no such entry.

**H1 queries:** As in the proof of Theorem 3.1, $A_2$ requests $\Omega_{ijkl} = (m_i||w_j||\text{ID}_k, V_i)$. $B$ returns $h_{1ijkl} \leftarrow \mathcal{G}_1$ to $A_2$ and adds $(\Omega_{ijkl}, h_{1ijkl})$ to the $H_1$-list if there is no such entry.

**H2 queries:** As in the proof of Theorem 3.1, $A_2$ requests $\Omega_{ijkl} = (m_i||w_j||\text{ID}_k, V_i)$. $B$ returns $h_{2ijkl} \leftarrow \mathcal{G}_1$ to $A_2$ and adds $(\Omega_{ijkl}, h_{2ijkl})$ to the $H_2$-list if there is no such entry.

**Key extract queries:** As in the proof of Theorem 3.1, $A_2$ requests $\text{ID}_i$. $B$ returns $s_{\text{ID}_i}$ to $A_2$ and adds ($\text{ID}_i, s_{\text{ID}_i}$) to the Key-list if there is no such entry.

**Delegation queries:** As in the proof of Theorem 3.1, $A_2$ requests $(w_i, X_j)$ where $|X_j| \leq q$. $B$ first computes $V_j$ with the help of $T$ and $X_j$, then $B$ computes

$$D = (H_0(w_i, V_j) \cdot V_j)^{sk_C} = h_{0ij}^{sk_C} \cdot V_j^{sk_C}$$

and returns it to $A_2$.

**Signature queries:** As in the proof of Theorem 3.1, $A_2$ requests $(m_i, w_j, \text{ID}_k, X_l)$ where $|X_l| \leq q$. $B$ first computes $V_l$ with the help of $T$ and $X_l$, then $B$ computes

$$\Sigma_{ijkl} = h_{0ij}^{sk_C} \cdot V_i^{sk_C} \cdot h_{1ijkl}^{s_{\text{ID}_k}} \cdot h_{2ijkl}^{r},$$

$$W_{\text{ID}_k} = g^{h_{1ijkl}^H(\text{ID}_k)(h+x)},$$

and $R = g^r$ where $r \leftarrow \mathbb{Z}_p^*$. Then $B$ returns $\sigma = (\Sigma_{ijkl}, W_{\text{ID}_k}, R)$ to $A_2$. 
3.4. Security Analysis

After all the queries, \( A_2 \) outputs a message-signature tuple \((m^*, w^*, X^*, \text{ID}^*, \sigma^*)\) with successful probability at least \( \epsilon \) where \( \text{ID}^* \) is the identity of a host which is not in the designated host list \( X^* \) and \( \sigma^* = (\Sigma^*, W^*, R^*) \) is a valid signature. We assume there exists tuples \((\text{ID}^*, s_{\text{ID}^*}), (\text{ID}^*, h^*), ((w^*, V^*), h_0^*), ((m^*||w^*||\text{ID}^*, V^*), h_1^*)\) and \(((m^*||w^*||\text{ID}^*, V^*), h_2^*)\) in the Key-list, \( H\)-list, \( H_0\)-list, \( H_1\)-list and \( H_2\)-list respectively. Here,

\[ V^* = g^{\prod_{h \in X^*(h+x)}}. \]

Therefore, it satisfies

\[
\hat{e}(\Sigma^*, g) = \hat{e}(H_0(w^*, V^*), pk_C) \cdot \hat{e}(W^*, pk_C^{H(\text{ID}^*)} \cdot E) \cdot \\
\hat{e}(H_1(m^*||w^*||\text{ID}^*, V^*), P_{\text{ID}^*}) \cdot \hat{e}(H_2(m^*||w^*||\text{ID}^*, V^*), R^*). 
\]

From this equation, we have

\[ (W^*)^{(h^*+x)} = V^* = g^{\prod_{h \in X^*(h+x)}} \]

and \( h^* \notin X^* \). \( (h^*, g^{\frac{1}{\sigma^*+x}}) \) can be computed with the help of \( T \) and \( X^* \) from the above equation, therefore the given instance of the \( q \)-SDH problem is solved. The probability of \( B \) to solve the given instance of the \( q \)-SDH problem is the same as for \( A_2 \) who breaks the proposed scheme. The total running time of \( B \) is at most \( t + c_{(G_1, G_2)}(q_{H^*} + q_D + q_S + q_{KE} + 1) \), which is similar to the proof of Theorem 3.1. This completes the proof.

### 3.4.3 Existential Unforgeable Against Adversary \( A_3 \)

**Theorem 3.3** If there exists an adversary \( A_3 \) who can \((t, q_{H^*}, q_S, \epsilon)\)-break the proposed scheme, then there exists another algorithm \( B \) that can use \( A_3 \) to solve an instance of the CDH problem in \( G_1 \) with probability

\[
\text{Succ}_{B,G_1}^{CDH} \geq (1 - \frac{2}{q_S + 2})^{q_S} (\frac{2}{q_S + 2})^2 \epsilon
\]

in time \( t + c_{(G_1, G_2)}(q_{H^*} + q_S + 1) \). Here \( c_{(G_1, G_2)} \) is a constant that depends on \((G_1, G_2)\).

**Proof.** The proof is similar to the proof of Theorem 3.1. It therefore is omitted.
3.5 Summary

In this chapter, we proposed a secure mobile agent that allows a mobile agent owner to select remote hosts for the designated agent network and eliminates the non-repudiation and misuse problems in the proxy-based mobile agent model. We also provided a rigorous security proof and showed that our scheme is proved secure against the strongest adversaries. We defined the security model which captures the most powerful attacks against adaptive-chosen-message and adaptive-chosen-host in the random oracle model. The communication overhead is an important issue in mobile agent applications. The signature length of the scheme presented in this chapter is three elements of the group $G_1$, which is shorter than most exiting schemes. In next chapter, we will improve this scheme and propose an even shorter and more efficiency scheme for mobile agent applications.
Secure Mobile Agents with Less Communication Overhead

The size of the mobile agents depends on the data they carry. Therefore, reducing the size of the carried data is a significant issue. In this chapter, we propose a novel scheme which reduces the communication overhead of mobile agents. The scheme is derived from the secure mobile agent scheme presented in Chapter 3.

In Section 4.1, we introduce the security model. As in the Chapter 3, we defined three different types of adversaries in our security model. Unlike the scheme presented in Chapter 3, the security of this scheme is based on the hardness of the Gap Bilinear Diffie-Hellman problem and the $q$-Strong Diffie-Hellman Problem, respectively. We then propose our new mobile agent scheme in Section 4.2. The security analysis of this scheme is presented in Section 4.3. As in the Chapter 3, we provide three theorems which prove the security of proposed scheme against three adversaries respectively. In Section 4.4, we present a comprehensive comparison of the bandwidth and the performance between proposed scheme and the scheme presented in Chapter 3. We summarise this chapter in Section 4.5.

4.1 Security Model

The security model of our mobile agent scheme is almost the same as the security model defined in Chapter 3. We now add queries to the signature verification oracle:

- **Verify queries**: adversary $\mathcal{A}$ can ask a signature verification query of $(m, w, ID, \sigma)$ with the message $m$, the warrant $w$, the identity $ID$, of the host who signs the service, where $\sigma$ is the signature that the adversary wants to verify.

  In response, the challenger $\mathcal{C}$ outputs **TRUE** if it is correct, or **FALSE** otherwise.

We define $\text{Succ}_\mathcal{A}$ to be the probability that the adversary $\mathcal{A}$ wins the game.
Definition 4.1 We say an adversary $A$ can $(t, q_{KE}, q_D, q_S, q_V, \epsilon)$-break our scheme if $A$ runs in time at most $t$, makes at most $q_{KE}$ key extract queries, at most $q_D$ delegation queries, at most $q_S$ signature queries, and at most $q_V$ verify queries, $\text{Succ}_A$ is at least $\epsilon$. Here $\mathcal{A} \in \{A_1, A_2, A_3\}$ where $A_1, A_2, A_3$ are the different types of the adversary defined in Chapter 3.

### 4.1.1 Complexity Assumptions

We recall the complexity definitions and the assumptions which will be used in our security proofs.

#### Definition 4.2 (Gap Bilinear Diffie-Hellman (GBDH) Problem.)
Given cyclic groups $G_1, G_2$ over a finite field, a generator $g$ of $G_1$, a bilinear map $\hat{e} : G_1 \times G_1 \rightarrow G_2$, and three elements $g^x, g^y, g^z \in G_1$, for some unknown $x, y, z \overset{\$}{\leftarrow} \mathbb{Z}_p$, compute $\hat{e}(g, g)^{xyz} \in G_2$ with the help of the DBDH oracle.

#### Definition 4.3 (Gap Bilinear Diffie-Hellman (GBDH) Assumption.)
Given cyclic groups $G_1, G_2$ over a finite field, a generator $g$ of $G_1$, a bilinear map $\hat{e} : G_1 \times G_1 \rightarrow G_2$, three elements $g^x, g^y, g^z \in G_1$, for some unknown $x, y, z \overset{\$}{\leftarrow} \mathbb{Z}_p$, and the DBDH oracle $O_{DBDH}$, the following function $\text{Succ}_{A, G_1, G_2}^{GBDH}$ is negligible for any polynomially bounded algorithm $A$.

$$\text{Succ}_{A, G_1, G_2}^{GBDH} = \text{Pr}[A(g, \hat{e}, g^x, g^y, g^z, O_{DBDH}) = \hat{e}(g, g)^{xyz} : x, y, z \overset{\$}{\leftarrow} \mathbb{Z}_p].$$

#### Definition 4.4 (q-Strong Diffie-Hellman (q-SDH) on $G_1$.)
Given a tuple $(g, g^s, \ldots, g^{s^q})$, for a unknown $s \overset{\$}{\leftarrow} \mathbb{Z}_p^*$ and $q \in \mathbb{Z}_p^*$, compute a pair $(c, g^{c+s^q})$ where $c \in \mathbb{Z}_p$.

#### Definition 4.5 (q-Strong Diffie-Hellman (q-SDH) Assumption on $G_1$.)
Given a tuple $(g, g^s, \ldots, g^{s^q})$, for a unknown $s \overset{\$}{\leftarrow} \mathbb{Z}_p^*$ and $q \in \mathbb{Z}_p^*$, the following function $\text{Succ}_{A, G_1}^{q-SDH}$ is negligible for any polynomially bounded algorithm $A$.

$$\text{Succ}_{A, G_1}^{q-SDH} = \text{Pr}[A(g, g^s, \ldots, g^{s^q}) = ((c, g^{1+c}) \land c \in \mathbb{Z}_p)].$$

### 4.2 Our Scheme

1. **Setup:** Let $(G_1, G_2)$ be cyclic multiplicative groups where $|G_1| = |G_2| = p$ for some prime $p$. Let $g$ be a generator of $G_1$. Select four distinct secure hash
functions: \( H, H_0, H_1, \) and \( H_2 \) where \( H : \{0, 1\}^* \rightarrow \mathbb{Z}_p^*, H_0 : \{0, 1\}^* \times G_1 \rightarrow G_1, \)
\( H_1 : \{0, 1\}^* \rightarrow G_1, \) and \( H_2 : G_2 \rightarrow \mathbb{Z}_p^*. \) The system parameter is \( \text{Param} = (\hat{\epsilon}, G_1, G_2, p, g, H, H_0, H_1, H_2). \)

2. Verifier Setup: Pick a random \( s_V \in \mathbb{Z}_p^* \) as its private key. Set the public key as \( pk_V = g^{s_V} \in G_1. \)

3. Customer Setup:
   - Select a random number \( s_C \in \mathbb{Z}_p^* \) as his private key and compute the corresponding public key \( pk_C = g^{s_C} \in G_1. \)
   - Randomly select \( t \in \mathbb{Z}_p^* \) and compute a tuple \( T = (g, g^t, g^{t^2}, g^{t^3}, \ldots, g^{t^\theta}) \), where \( \theta \) is the upper bound. e.g. the customer can only build a designated hosts list that includes \( k \) hosts where \( k \leq \theta. \)
   - Build the designated host list:
     - The customer chooses \( k \) hosts, \( ID_1, ID_2, \ldots, ID_k \), and computes \( X = (H(ID_1), H(ID_2), \ldots, H(ID_k)) \subset \mathbb{Z}_p \setminus \{-t\} \) where \( ID_i \) is the identity of a host and \( k \leq \theta. \)
     - Compute
       \[
       V = g^{\prod_{i=1}^{k}(H(ID_i) + t)} \in G_1.
       \]
   - Publish \( PK_C = (T, pk_C, X, V) \) as the customer’s public key and keep \( s_C, t \) as his private keys.

4. Agent Setup:
   - Compute the delegation token
     \[ D = H_0(w, pk_V)^{s_C} \]
     where \( w \in \{0, 1\}^* \) is his warrant, \( pk_V \) is the verifier’s public key.
   - The customer embeds \( (w, D) \) in the mobile agent.

5. Host Execution:
   - Select a random number \( s_i \in \mathbb{Z}_p^* \) as his private key where \( ID_i \) is his identity. The public key of this host is \( pk_i = g^{s_i} \in G_1. \)
4.3 Security Analysis

- Verify the delegation token by checking

\[ \hat{e}(D, g) \overset{?}{=} \hat{e}(H_0(w, pk_V), pk_C). \]

- If invalid, the host stops the execution.
- Otherwise, the host computes \( H(\text{ID}_i) \), if \( H(\text{ID}_i) \notin X \), the host stops the execution.
- Otherwise, if \( H(\text{ID}_i) \in X \), the host computes

\[ \Sigma = H_2(\hat{e}(D \cdot H_1(m||w||\text{ID}_i)^{\epsilon}, pk_V)) \]

and

\[ W_i = g^{H_0(\text{ID}_i)}(h+t). \]

Here, \( W_i \) can be computed using \( T \) and \( X \) without \( t \). \( m \in \{0,1\}^* \) is the message (the offer of the host for the transaction) needs to be signed. The host outputs

\[ \sigma = (\Sigma, W_i) \]

as the signature of the message \( m \) under the delegation \( D \).

6. Verification: Given \( \text{Param} \), the verifier’s public key \( pk_V \), the customer’s public key \( (T, pk_C, X, V) \), the host’s public key \( pk_i \), the host’s identity \( \text{ID}_i \), a warrant \( w \), a message \( m \), and a signature \( \sigma = (\Sigma, W_i) \), only the designated verifier can verify the validity of the signature by checking

\[
\begin{aligned}
\Sigma & \overset{?}{=} H_2(\hat{e}(H_0(w, pk_V)^{\epsilon}, pk_C) \cdot \hat{e}(H_1(m||w||\text{ID}_i)^{\epsilon}, pk_i)), \\
\hat{e}(W_i, g^{H(\text{ID}_i)} \cdot g^t) & \overset{?}{=} \hat{e}(V, g).
\end{aligned}
\]

If the above equations hold, \( \sigma \) will be accepted as a valid signed service; otherwise, rejected.

4.3 Security Analysis

4.3.1 Existential Unforgeable Against Malicious Host within Designated List

Theorem 4.1 Suppose there exists an adversary \( A_1 \) (the malicious host within designated list) who can \((t, q_{H_0}, \epsilon)\)-break the proposed scheme. \( q_{H_0} \) represents \( A_1 \)
can make at most $q_{H_0}$ queries to the oracle $H_0$. Then, there is an algorithm $\mathcal{B}$ who can use $\mathcal{A}_1$ to solve an instance of the GBDH problem in $\mathbb{G}_1$ with probability $\text{Succ}_G^{\text{GBDH}} = \epsilon/(eq_{H_0})$ in time $O(\text{time}(\mathcal{A}_1))$.

Proof: Algorithm $\mathcal{B}$ is given a random instance $(g, g^a, g^b, g^c)$ of Gap Bilinear Diffie-Hellman (GBDH) problem in $\mathbb{G}_1$. Its goal is to compute $\hat{c}(g, g)^{abc}$ by interacting with adversary $\mathcal{A}_1$ with the help of Decisional Bilinear Diffie-Hellman (DBDH) Oracle. Let $\Omega = \hat{c}(g, g)^{abc}$ be the solution to this instance of the GBDH problem. $\mathcal{B}$ simulates the challenger and interacts with $\mathcal{A}_1$ as described below. We assume that $\mathcal{A}_1$ is well-behaved in the sense that $\mathcal{A}_1$ will never repeat the same queries in the simulation.

Setup: As in Theorem 3.1 except set the customer’s public key $pk_C = g^a$, the verifier’s public key $pk_V = g^c$. Return $\text{Param}, PK_C = (T, pk_C, X, V), pk_V$ to $\mathcal{A}_1$. $\mathcal{B}$ randomly picks a number $\delta \leftarrow \{1, 2, ..., q_{H_0}\}$.

H-queries: Same as in Theorem 3.1.

$H_0$-queries: $\mathcal{A}_1$ makes queries to the random oracle $H_0$ at any time. To respond to these queries, $\mathcal{B}$ maintains an $H_0$-list of tuples $(w_i, Q_i, b_i, i)$. The list is initially empty. Here $i$ represents the sequence of the queries ($1 \leq i \leq q_{H_0}$). For each query $w_i$, $\mathcal{B}$ responds as follows:

1. If $w_i$ appears in a tuple $(w_i, Q_i, b_i, i)$ of the $H_0$-list, $\mathcal{B}$ responds to $\mathcal{A}$ with $Q_i$.
2. Otherwise, if $i = \delta$, set $Q_\delta = g^b$ and add $(w_\delta, Q_\delta, -, \delta)$ to the $H_0$-list. $\mathcal{B}$ responds to $\mathcal{A}$ with $Q_\delta$.
3. If $i \neq \delta$, pick a random $b_i \leftarrow \mathbb{Z}_p^*$ and compute $Q_i = g^{b_i}$. Add $(w_i, Q_i, b_i, i)$ to the $H_0$-list. $\mathcal{B}$ responds to $\mathcal{A}$ with $Q_i$.

$H_1$-queries: $\mathcal{A}_1$ can make queries to the $H_1$ oracle of the input $x_{ijk}$ where $x_{ijk} = m_i || w_j || \Sigma_i$. For each query $x_{ijk}$, $\mathcal{B}$ maintains an $H_1$-list of tuples $(x_{ijk}, h_{ijk}^1)$ and responds to $\mathcal{A}_1$ as follows:

1. If $x_{ijk}$ appears in a tuple $(x_{ijk}, h_{ijk}^1)$ of the $H_1$-list, $\mathcal{B}$ responds to $\mathcal{A}$ with $h_{ijk}^1$.
2. Otherwise, pick a random $h \leftarrow \mathbb{G}_1$ where there is no item $(., h)$ in the $H_1$-list. Add $(x_{ijk}, h)$ to the $H_1$-list. $\mathcal{B}$ responds to $\mathcal{A}$ with $h$.

$H_2$-queries: $\mathcal{A}_1$ can make queries to the $H_2$ oracle of the input $y_i \in \mathbb{G}_2$ at any time. To respond to these queries, $\mathcal{B}$ maintains two lists: $H_2^1$-list of tuples $(y_i, \Sigma_i)$ and $H_2^2$-list of tuples $(\perp, z_i, \Sigma_i)$. For each $\Sigma_i$, it appears either in the $H_2^1$-list or in
the $H_2^2$-list. In other words, if $\Sigma_i$ appears in a tuple $(\cdot, \Sigma_i)$ on the $H_2^1$-list, then $\Sigma_i$ must not be in a tuple $(\perp, \cdot, \Sigma_i)$ on the $H_2^2$-list. Both of the lists are initially empty.

For each query $y_i$, $B$ responds to $A_1$ as follows:

1. If the query $y_i$ already appears on the $H_2^1$-list in a tuple $(y_i, \Sigma_i)$, $B$ returns to $A_1$ with $\Sigma_i$.

2. Otherwise, $B$ checks the items on the $H_2^2$-list by submitting $y_i/z_i$ to the DBDH oracle for each $z_i$. If there exists a tuple $(\perp, z^*, \Sigma^*)$ such that the query $y_i/z^*$ to the DBDH oracle can make the DBDH oracle output $\text{TRUE}$. Therefore, the given instance of the GBDH problem is solved. $B$ returns to $A_1$ with $\Sigma^*$.

3. Otherwise, $B$ picks a random $\Sigma \in \mathbb{Z}_p^*$ such that there is no item $(\cdot, \Sigma)$ on the $H_2^1$-list. $B$ adds $(y_i, \Sigma)$ into the $H_2^1$-list and returns to $A_1$ with $\Sigma$.

**Key extract queries:** Same as in Theorem 3.1.

**Delegation queries:** Similar as in Theorem 3.1. For each query $w_i$, if $Q_i = H_0(w_i, pk_V) = g^b$, $B$ reports failure and terminates. Otherwise, $B$ returns $(g^a)^{b_i}$ to $A_1$.

**Signature queries:** For a query $(m_i, w_j, \text{ID}_k)$ where $m_i$ is the message to be signed, $w_j$ is the customer’s warrant, $\text{ID}_k$ is the host’s identity. Suppose there exists a tuple $(w_j, Q_j, b_j, j)$ on the $H_0$-list, a tuple $(x_{ijk}, h_{ijk}^1)$ where $x_{ijk} = m_i||w_j||\text{ID}_k$ on the $H_1$-list. Compute

$$W_k = g^{\prod_{x \in X} h_{ijk}(\text{ID}_k)(h+1)}.$$  

1. If $j \neq \delta$, $Q_j = g^b$. Set

$$y^* = \hat{e}(g_{ij}^a \cdot (h_{ijk}^1)^{s_k}, pk_V) = \hat{e}((g^a)^b \cdot (h_{ijk}^1)^{s_k}, g^c),$$

where $s_k$ is the private key of host $\text{ID}_k$. $B$ then checks the $H_2^1$-list. If there exists a tuple $(y^*, \Sigma^*)$ on the $H_2^1$-list, return $(\Sigma^*, W_k)$ to $A_1$. Otherwise, pick a random $\Sigma \in \mathbb{Z}_p^*$ and add $(y^*, \Sigma)$ into the $H_2^1$-list. Return $(\Sigma, W_k)$ to $A_1$.

2. Otherwise, if $j = \delta$ hence $Q_j = g^b$. Set

$$y^* = \hat{e}(g_{ij}^{ab} \cdot (h_{ijk}^1)^{s_k}, pk_V) = \hat{e}(g_{ij}^{ab}, g^c) \cdot \hat{e}((h_{ijk}^1)^{s_k}, g^c) = \Omega \cdot \hat{e}((h_{ijk}^1)^{s_k}, g^c).$$

As $\Omega$ is unknown to $B$ therefore $y^*$ is unknown to $B$. Set $z^* = \hat{e}((h_{ijk}^1)^{s_k}, g^c)$.

- If there exists a tuple $(\perp, z^*, \Sigma^*)$ on the $H_2^1$-list, $B$ returns $(\Sigma^*, W_k)$ to $A_1$ as the answer of the signature query.
4.3. Security Analysis

- Otherwise, $\mathcal{B}$ checks the tuples $(y_i, )$ on the $H_1^2$-list by submitting $y_i/z^*$ to the DBDH oracle for each $y_i$ on the $H_1^2$-list. If there exists a tuple $(y, \Sigma)$ such that the query $y/z^*$ to the DBDH oracle can make the DBDH oracle outputs $\text{TRUE}$. Therefore, the given instance of the GBDH problem is solved. $\mathcal{B}$ returns to $\mathcal{A}_1$ with $(\Sigma, W_k)$.

- Otherwise, pick a random $\Sigma \in \mathbb{Z}_p^*$ and add $(\bot, z^*, \Sigma)$ to the $H_2^2$-list. $\mathcal{B}$ returns to $\mathcal{A}_1$ with $(\Sigma, W_k)$.

**Verify queries:** For a query $(m_i, w_j, \text{ID}_k, \sigma_{ijk})$ where $\sigma_{ijk} = (\Sigma_{ijk}, W_k)$, $\mathcal{B}$ checks:

1. If $\hat{e}(W_k, g^{H(\text{ID}_k)} \cdot g^t) \neq \hat{e}(V, g)$, $\mathcal{B}$ rejects it as an invalid signature.

2. Otherwise, $\hat{e}(W_k, g^{H(\text{ID}_k)} \cdot g^t) = \hat{e}(V, g)$. Suppose there exists a tuple $(w_j, Q_j, b_j, j)$ on the $H_0$-list, a tuple $(x_{ijk}, h^1_{ijk})$ where $x_{ijk} = m_i || w_j || \text{ID}_k$ on the $H_1$-list.

   - if $j \neq \delta$, $Q_j = g^{b_j}$. Set
     
     $$y^* = \hat{e}((g^{b_j})^a \cdot (h^1_{ijk})^{s_k}, pk_V) = \hat{e}((g^a)^{b_j} \cdot (h^1_{ijk})^{s_k}, g^c).$$

     If there exists a tuple $(y^*, \Sigma_{ijk})$ on the $H_3^2$-list, $\mathcal{B}$ will accept it as a valid signature. Otherwise, reject it as an invalid signature.

   - If $j = \delta$, $Q_j = g^b$. Set $z^* = \hat{e}((h^1_{ijk})^{s_k}, g^c)$. If there exists $(\bot, z^*, \Sigma_{ijk})$ on the $H_2^2$-list, $\mathcal{B}$ will accept it as a valid signature.

   - $\mathcal{B}$ checks the tuples $(y_i, )$ on the $H_2$-list by submitting $y_i/z^*$ to the DBDH oracle for each $y_i$ on the $H_1^2$-list. If there exists a tuple $(y, \Sigma_{ijk})$ such that the query $y/z^*$ to the DBDH oracle can make the DBDH oracle output $\text{TRUE}$. $\mathcal{B}$ will accept it as a valid signature.

   - Otherwise, $\mathcal{B}$ rejects it as an invalid signature.

Since $H_2$ is uniformly distributed, it has only negligible probability that $(m_i, w_j, \text{ID}_k, \Sigma_{ijk})$ is a valid signature while $\Sigma_{ijk}$ is not queried from oracle $H_2$.

**Output:** $\mathcal{A}_1$ outputs a valid signature $(m^*, w^*, \text{ID}^*, \sigma^*)$. Set $\sigma^* = (\Sigma^*, W^*)$. Therefore, there must exist a tuple $(\bot, \Sigma^*)$ on the $H_3^2$-list (since $H_2$ is uniformly distributed). Suppose it is $(y^*, \Sigma^*)$.

1. If $w^* \neq w_\delta$, $\mathcal{B}$ reports failure and terminates.
2. Otherwise, $\mathcal{B}$ computes
\[ \hat{e}(g, g)^{abc} = \frac{y^*}{\hat{e}((h^1_*)^{s^*}, g^c)}, \]
where $s^*$ is the private key of the host $\text{ID}^*$, $(m^*||w^*||\text{ID}^*, h^1_*)$ is the tuple on the $H_1$-list.

Analysis: If $w^* = w_\delta$, therefore in order to output a valid signature, $\mathcal{A}_1$ has to compute
\[ y^* = \hat{e}(D \cdot H_1(m^*||w^*||\text{ID}^*)^{s^*}, pk_V) = \hat{e}(D, pk_V) \cdot \hat{e}((h^1_*)^{s^*}, pk_V) = \hat{e}(g^{ab}, g^c) \cdot \hat{e}((h^1_*)^{s^*}, g^c). \]
So $\mathcal{B}$ can compute
\[ \hat{e}(g, g)^{abc} = \frac{y^*}{\hat{e}((h^1_*)^{s^*}, g^c)} \]
and hence solve the given instance of the GBDH problem. The probability that $\mathcal{B}$ solves the given instance of the GBDH problem depends on the following events:

- $\mathcal{B}$ does not abort during the delegation queries. The probability is at least 
  \[ (1 - 1/q_{H_0})^{q_D} \geq 1/\epsilon. \]
- $\mathcal{B}$ does not abort during the output phase. The probability is at least $1/q_{H_0}$.
- $\mathcal{A}_1$ outputs a valid signature where the probability is at least $\epsilon$.

Therefore, $\mathcal{B}$ can successfully solve the given instance of the GBDH problem with probability
\[ \text{Succ}_{\mathcal{B}}^{\text{GBDH}} = \frac{\epsilon}{e q_{H_0}}. \]

### 4.3.2 Existential Unforgeable Against Malicious Host without in Designated List

**Theorem 4.2** Suppose there exists an adversary $\mathcal{A}_2$ (the malicious host without in designated list) who can $(t, q_{H^*}, q_{KE}, q_D, q_S, q_V, \epsilon)$-break the proposed scheme. $q_{H^*}$, $q_{KE}$, $q_D$, $q_S$, $q_V$ represents $\mathcal{A}_2$ can make at most total $q_{H^*}$ queries to all the hash oracles, at most $q_{KE}$ key extract queries, at most $q_D$ delegation queries, at most $q_S$ signature queries, and at most $q_V$ verification queries respectively. Then there is an algorithm $\mathcal{B}$ who can use $\mathcal{A}_2$ to solve an instance of the $q$-SDH problem in $\mathbb{G}_1$ with probability $\text{Succ}_{\mathcal{B}}^{q^{*}\text{SDH}} = \text{Succ}_{\mathcal{A}_2} \geq \epsilon$ in time $O(\text{time}(\mathcal{A}_2))$.

**Proof:** The proof is similar to the proof of Theorem 3.2. It therefore is omitted.
4.4. Comparison

4.3.3 Existential Unforgeable Against Malicious Customer

**Theorem 4.3** Suppose there exists an adversary $A_3$ (the malicious customer) who can $(t, q_{H_0}, \epsilon)$-break the proposed scheme. $q_{H_0}$ represents $A_3$ can make at most $q_{H_0}$ queries to the oracle $H_0$. Then there is an algorithm $B$ who can use $A_3$ to solve an instance of the GBDH problem in $G_1$ with probability $\text{Succ}_{G}^{\text{GBDH}} = \epsilon/q_{H_0}$ in time $O(\text{time}(A_3))$.

**Proof:** The proof is similar to the proof of Theorem 4.1. It therefore is omitted.

4.4 Comparison

In this section, we compare the signature length and the computation cost between the proposed scheme and the secure mobile agent scheme presented in Chapter 3. We do not compare the size of the delegation tokens because they are the same in both schemes. The computation cost of both schemes includes the signing cost and the verification cost.

**Bandwidth**

In the secure mobile agent scheme presented in Chapter 3, the signature is a tuple of three elements $(\Sigma, W, R)$ where they all belong to the group $G_1$. In the proposed scheme, the signature is a tuple of two elements $(\Sigma, W)$ where $\Sigma \in \mathbb{Z}_p$ and $W \in G_1$. Let $|\mathbb{Z}_p|$ denote the bit length of the element in $\mathbb{Z}_p$ and $|G_1|$ denote the bit length of the element in $G_1$. We have the following table 4.1:

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Signature Length</th>
<th>$p:160$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheme in Chapter 3</td>
<td>3$</td>
<td>G_1</td>
</tr>
<tr>
<td>Proposed Scheme</td>
<td>$</td>
<td>G_1</td>
</tr>
</tbody>
</table>

Table 4.1: Bandwidth Comparison Between Two Schemes.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>$G_1$(multi.)</th>
<th>$G_1$(add.)</th>
<th>$G_2$</th>
<th>Pairings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheme in Chapter 3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>(Sign + Verify)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proposed Scheme</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>(Sign + Verify)</td>
<td></td>
<td></td>
<td></td>
<td>(1 can be pre-computed)</td>
</tr>
</tbody>
</table>

Table 4.2: Performance Comparison Between Two Schemes.
4.5 Summary

In this chapter, we improved the scheme presented in Chapter 3 and proposed a novel secure mobile agent scheme. This scheme possesses all the security properties that the previous scheme has. In addition, this scheme offers a shorter signature size and better efficiency, in comparison with the previous scheme presented in Chapter 3. A comprehensive comparison of bandwidth and performance is given. Generally speaking, the signature length of this scheme is reduced one-third compared to the previously proposed scheme. We also defined the security model and provided a rigorous proof to this scheme.
In this chapter, we propose a novel secure scheme for information exchanges in multi-agent systems. Our scheme offers both security and privacy protection. We are motivated by some useful features of Oblivious Signature-Based Envelopes (OSBE) and hidden credentials to construct our scheme. Attributes are imported into our scheme to distinguish the different affiliations of agents. Each agent in the multi-agent system may or may not possess some attributes which are issued and maintained by the group manager of the multi-agent system. When two agents start a negotiation, attributes are used to authenticate each other where one agent must satisfy some attribute policies required by the other agent in order to pass its authentication. The attribute-based authentication mechanism can prevent the negotiation messages and the attributes of the negotiation agents from being revealed to unexpected parties.

In Section 5.1, we provide a high-level introduction to the multi-agent system, followed by the protocol and the security properties. We also provide some preliminaries and the complexity assumptions. In Section 5.2, we formally present our scheme. In Section 5.3, we present the security proofs to our scheme. We summarise this chapter in Section 5.4.

5.1 Model and Definitions

5.1.1 A High-Level Description of Our Scheme

We assume that agents reside in different hosts in a networked environment and are grouped by a group manager (a trusted party) who issues credentials and private keys
to agents. The group manager could be the Privacy Key Generator in an identity-based system. The group manager maintains two lists: (1) a list of pseudonyms for the agents that are registered with the group, and (2) a list of policy attributes. The group manager issues a credential embedded with some suitable policy attributes to an agent that is registered with the group. When an agent sends a message to other agents, it encrypts the message with the selected policy attributes associated with the receivers’ pseudonyms, so that only the agents that hold the same policy attributes can open the message.

Figure 5.1: Attribute-based Information Exchange in the Multi-agent System

Figure 5.1 illustrates how our protocol works, where five agents, labeled from 1 to 5, reside in five corresponding hosts. All agents except agent 5 are registered with the trusted group manager and thus form a multi-agent group. For each registered member, the group manager issues a pseudonym, a private key, and the credentials with respect to certain policy attributes. The group manager maintains a member list which stores pseudonyms and an attribute list. Agent 5 is not registered; therefore it is not recorded in the list. Suppose that agent 2 sends a message which is
5.1. Model and Definitions

encrypted using its selected policy attributes. Only the agents that possess the credentials with the matching policy attributes are able to open this encrypted message and hence authenticate the sender. In this example, we assume that agents 3 and 4 hold the correct credentials respectively. Although agent 1 has registered with the group, it does not hold the matching policy attributes which satisfy the encryption policy of the message and therefore cannot open the encrypted message and learns nothing about the sender. Obviously, agent 5 is out of the picture, since it is not registered with the group.

5.1.2 Outline of Our Scheme

Our scheme consists of the following algorithms: Setup, RegisterAgent, RevokeAgent, IssueAttribute, SendMsg and RcvMsg. We define the policy attributes in a disjunctive normal form (DNF) or a conjunctive normal form (CNF).

1. Setup. Selecting as input a security parameter $k$, this algorithm outputs the system parameters $Params$ and the master key $s$. That is,

$$(Params, s) \leftarrow \text{Setup}(1^k).$$

2. RegisterAgent. For an agent with identity $ID_i$, the group manager (GM) runs this algorithm to register this agent by issuing a pair $(h_i, d_i)$, where $h_i$ is the pseudonym of $ID_i$ and $d_i$ is the private key. That is,

$$(h_i, d_i) \leftarrow \text{RegisterAgent}(Params, s, ID_i).$$

GM adds $h_i$ into the group member list $L_m$ if it has not been entered.

3. RevokeAgent. GM runs this algorithm to revoke a group member by removing its information from the group member list $L_m$.

4. IssueAttribute. GM runs this algorithm to issue a credential $Cred_i$ with respect to attribute $Atb_i$. That is,

$$Cred_i \leftarrow \text{IssueAttribute}(Params, s, Atb_i).$$

GM adds $Atb_i$ into the attribute list $L_a$ if it has not been entered.
5. **SendMsg.** Taking as input the system parameter $Params$, a sender’s private key $d_i$, a policy $POL$ (in DNF) the sender chooses and the message $M$, this algorithm outputs an encrypted message $C$. That is,

$$C \leftarrow \text{SendMsg}(Params, d_i, POL, M).$$

Here, $POL = \bigvee_{k=1}^{l}(POL_k)$. $POL_k$ is the concatenation of any pseudonym and any number of the attributes the sender chooses such as $POL_k = h_1 \bigwedge_{j=1}^{v'} (Atb_j)$.

6. **RcvMsg.** Taking as input the system parameter $Params$, the private key $d_i$, the corresponding policy attributes $\{\text{Cred}_1, ..., \text{Cred}_v\}$, the encrypted message $C$, this algorithm outputs a pair $(M, h)$ where $M$ is the plaintext of $C$ and $h$ is the sender’s pseudonym, if and only if $\{\text{Cred}_1, ..., \text{Cred}_v\}$ matches the policy attributes embedded in $C$. That is,

$$(M, h) \leftarrow \text{RcvMsg}(Params, d_i, \{\text{Cred}_j\}_{j=1}^{v'}, C).$$

### 5.1.3 Security Definitions

In this section, we present the security properties of our scheme.

**Adaptive Chosen Ciphertext Secure.**

This property is a variant of the chosen ciphertext security (IND-CCA). Apart from making queries to the decryption oracle as in IND-CCA, the adversary in our model is allowed to make queries of private keys and credentials except the one that is being challenged. We define an adaptive chosen ciphertext attacker in our scheme as $A_1$. We want to capture the following attack scenarios:

- $A_1$ can query private keys and credentials he adaptively chooses except the target one.
- $A_1$ can make decryption queries to the oracle adaptively except the challenged one.

We define the following game between a randomized and polynomially bounded adversary $A_1$ and a challenger $C$: 
• **Setup:** The challenger $C$ runs the algorithm $\text{Setup}$ with the security parameter $k$ to obtain the system parameters $\text{Params}$ and the master key. It returns $\text{Params}$ to $A_1$ and keeps the master key to itself.

• **Phase 1:**
  
  - RegisterAgent/IssueAttribute queries: $A_1$ can ask $\text{ID}_i$ or $\text{Atb}_i$ he chooses adaptively. For an $\text{ID}_i$ query, $C$ runs the algorithm $\text{RegisterAgent}$ to obtain $(h_i, d_i)$ and responds to $A_1$ with $(h_i, d_i)$. For an $\text{Atb}_i$ query, $C$ runs the algorithm $\text{IssueAttribute}$ to obtain the credential $\text{Cred}_i$ of $\text{Atb}_i$ and responds to $A_1$ with $\text{Cred}_i$.
  
  - Decryption queries: $A_1$ can ask $(\text{POL}_i, C_i)$ he chooses adaptively. $C$ first runs the algorithm $\text{RegisterAgent}$ and $\text{IssueAttribute}$ to obtain the private key and credentials (denoted by $\text{CRED}_i$ for simplicity) of $\text{POL}_i$. $C$ then runs the algorithm $\text{RcvMsg}$ using $(\text{CRED}_i, C_i)$ to obtain $(M_i, h_i)$ and responds to $A_1$ with $(M_i, h_i)$.

• **Challenge:** Once $A_1$ decides that Phase 1 is over, $A_1$ outputs two equal length messages $M_0, M_1$, a sender’s pseudonym $h_S$ and a policy $\text{POL}$ he chooses to be challenged. Here, $\text{POL}$ is the concatenation of one pseudonym (of a receiver) and any number of attributes. The constraint is that at least one of the attributes or the pseudonyms in $\text{POL}$ has not been submitted in RegisterAgent/IssueAttribute queries in Phase 1. $C$ first runs the algorithm $\text{RegisterAgent}$ to obtain the private key $d_S$ of $h_S$. $C$ then picks a random bit $b \in \{0, 1\}$ and runs the algorithm $\text{SendMsg}$ using $(d_S, \text{POL}, M_b)$ to obtain the ciphertext $C$. It then sends $C$ as the challenge to $A_1$.

• **Phase 2:** Upon receiving the challenging ciphertext $C$, $A_1$ can still make RegisterAgent or IssueAttribute queries and Decryption queries adaptively as in Phase 1 except that
  
  1. At least one of the attributes or the pseudonyms in $\text{POL}$ must not be submitted in RegisterAgent/IssueAttribute queries.
  
  2. $(\text{POL}, C)$ must not be submitted in Decryption queries.

• **Guess:** Finally, $A_1$ outputs a guess $b' \in \{0, 1\}$ and wins the game if $b = b'$.
We define the advantage of such an adversary $A_1$ has in the above game as

$$\text{Adv-}A_1 = |\Pr[b = b'] - 1/2|.$$

**Definition 5.1** We say that our scheme is adaptive chosen ciphertext secure against a $(t, q_C, q_D)$ adaptive chosen ciphertext adversary $A_1$ if $\text{Adv-}A_1$ is negligible after making at most $q_C$ RegisterAgent/IssueAttribute queries and $q_D$ Decryption queries in time $t$.

**Impersonation Resistance.**

The adversary has only negligible probability in impersonating a target group member to interact with the other group members successfully, even if the adversary already corrupts some group members other than the target. We define an impersonation adversary in our scheme as $A_2$. We want to capture the attack scenarios as follows:

- $A_2$ can query private keys and credentials he adaptively chooses except the target one.
- $A_2$ can make encryption queries to the oracle adaptively except the challenged one.

We define the following game between a randomized and polynomially bounded adversary $A_2$ and a challenger $C$. Phase 1 of this game is identical to the Setup of the chosen ciphertext secure game. We only describe Phases 2, 3, 4 in the following:

- **Phase 2:** $A_2$ chooses target pseudonyms $h_S, h_R$, where he impersonates $h_S$ to convince $h_R$.

- **Phase 3:**
  - RegisterAgent/IssueAttribute queries: Same as the chosen ciphertext secure game. $ID_S, ID_R$ (corresponding to $h_S, h_R$ respectively) must not be queried.
  - Encryption queries: $A_2$ can ask $(h_i, POL_i, M_i)$ adaptively. $C$ first runs the algorithm RegisterAgent to obtain the private key $d_i$ of $h_i$. $C$ then runs the algorithm SendMsg using $(d_i, POL_i, M_i)$ to obtain the ciphertext $C_i$ and responds to $A_2$ with $C_i$. Here $POL_i = \bigvee_{k=1}^l (POL_k)$ and each $POL_k$
is the concatenation of one pseudonym (of a receiver) and any number of attributes. The constraint is \(h_i \neq h_S\) and \(h_R \notin \mathcal{POL}_k\) for all the \(\mathcal{POL}_k\).

- **Phase 4:** \(A_2\) outputs a ciphertext \(C\) under a policy \(\mathcal{POL}\) and wins the game if:
  1. \(h_R \in \mathcal{POL}\).
  2. \((M, h) \leftarrow \text{RcvMsg}(\text{Params}, \text{CRED}, C)\) where \(h = h_S\). \(\text{CRED}\) are the credentials that satisfy the policy \(\mathcal{POL}\).

We define the successful probability of such an impersonation adversary \(A_2\) wins the above game as \(\text{Succ-}A_2\).

**Definition 5.2** *We say that our scheme is impersonation resistant against a \((t, q_C, q_E)\) impersonation adversary \(A_2\) if \(\text{Succ-}A_2\) is negligible after making at most \(q_C\) RegisterAgent/IssueAttribute queries and \(q_E\) Encryption queries in time \(t\).*

**Probing Resistance.**

Adversary has only negligible probability to find whether an encrypted message \(C\) is valid, if the adversary does not hold the policy attributes embedded in \(C\) (even if the adversary is a valid group member). We define a probing adversary in our scheme as \(A_3\). We want to capture the following attack scenarios:

- \(A_3\) can query private keys and credentials adaptively except the target one.
- \(A_3\) can make encryption queries to the oracle adaptively.
- At least one of the attributes in the challenged policy has not been queried as one of the RegisterAgent/IssueAttribute queries.

We define the following game between a randomized and polynomially bounded adversary \(A_3\) and a challenger \(C\), Phase 1 of this game is identical to the Setup of chosen ciphertext secure game. We only describe Phase 2, 3, 4, 5, 6 in the following:

- **Phase 2:** \(A_3\) chooses a target sender \(h_S\), a policy \(\mathcal{POL}\) in CNF where \(h_{A_3} \in \mathcal{POL}\) and a message \(M\). Suppose \(A_3\) is in the group with pseudonym \(h_{A_3}\).
- **Phase 3:**
5.1. Model and Definitions

- RegisterAgent/IssueAttribute queries: Same as the chosen ciphertext secure game. \( \text{ID}_S \) (corresponding to \( h_S \)) must not be queried. At least one attribute in \( \mathcal{POL} \) must not be queried.

- Encryption queries: Same as the impersonation resistance game without constraint.

- **Phase 4:** \( C \) tosses a coin \( b \leftarrow \{0, 1\} \).

- **Phase 5:** If \( b = 0 \), \( C \) first runs the algorithm RegisterAgent to obtain the private key \( d_S \) of \( h_S \). \( C \) then runs the algorithm SendMsg using \( (d_S, \mathcal{POL}, M) \) to obtain the ciphertext \( C \). If \( b = 1 \), \( C \) runs a random simulation \( R \) to obtain a \( C \). After that \( C \) responds to \( A_3 \) with \( C \).

- **Phase 6:** \( A_3 \) outputs a guess \( b' \in \{0, 1\} \) and wins the game if \( b = b' \).

We define the advantage of such a probing adversary \( A_3 \) has in the above game as

\[
\text{Adv-} A_3 = |\Pr[b = b'] - 1/2|.
\]

**Definition 5.3** We say that our scheme is probing resistant against a \((t, q_C, q_E)\) probing adversary \( A_3 \) if \( \text{Adv-} A_3 \) is negligible after making at most \( q_C \) RegisterAgent/IssueAttribute queries and \( q_E \) Encryption queries in time \( t \).

**Indistinguishability to Eavesdroppers.**

For any encrypted message (an information exchange between some group members), the adversary has only negligible probability in distinguishing it from a simulation if the adversary is not involved in this information exchange. We define a distinguishable adversary in our scheme as \( A_4 \). We want to capture the following attack scenarios:

- \( A_4 \) can query private keys and credentials adaptively except the targets.
- \( A_4 \) can make encryption queries to the oracle adaptively.
- \( A_4 \) is not involved in the information exchange which is challenged.

We define the following game between a randomized and polynomially bounded adversary \( A_4 \) and a challenger \( C \), Phases 1, 4, 5, 6 of this game are identical to the probing resistance game. We only describe Phases 2, 3 in the following:
• Phase 2: $A_4$ chooses target sender $h_S$, receiver $h_R$, a policy $\mathcal{POL}$ in CNF where $h_R \in \mathcal{POL}$ and a message $M$.

• Phase 3:
  - RegisterAgent/IssueAttribute queries: Same as the impersonation resistance game.
  - Encryption queries: Same as the probing resistance game.

We define the advantage of such a detection adversary $A_4$ has in the above game as

$$\text{Adv}_{A_4} = |\Pr[b = b'] - 1/2|.$$ 

**Definition 5.4** We say that our scheme is indistinguishable to eavesdroppers against a $(t, q_C, q_E)$ distinguishable adversary $A_4$ if $\text{Adv}_{A_4}$ is negligible after making at most $q_C$ RegisterAgent or IssueAttribute queries and $q_E$ Encryption queries in time $t$.

**Hidden Credentials.**

The adversary has only negligible probability of finding which attributes are used in the encryption of a message, if he is not a candidature receiver. We define an adversary, who intends to find policy attributes from an encrypted message, as $A_5$.

We want to capture the following attack scenarios:

1. $A_5$ can choose the target policies he wants to challenge.
2. $A_5$ can query private keys and credentials adaptively except for the target one.
3. $A_5$ can make encryption queries to the oracle adaptively.

We define the following game between a randomized and polynomially bounded adversary $A_5$ and a challenger $C$, Phase 1, 4, 5, 6 of this game are identical to the probing resistance game. We only describe Phase 2, 3 in the following:

• Phase 2: $A_5$ chooses a target sender $h_S$, a policy $\mathcal{POL}$ in CNF and a message $M$.

• Phase 3:
  - RegisterAgent/IssueAttribute queries: Same as the chosen ciphertext secure game except that at least one attribute in $\mathcal{POL}$ must not be queried.
– Encryption queries: Same as the impersonation resistance game without constraint.

We define the advantage of such an adversary $\mathcal{A}_5$ has in the above game as

$$\text{Adv}_{\mathcal{A}_5} = |\Pr[b = b'] - 1/2|.$$ 

**Definition 5.5** We say that our scheme meets the property of hidden credentials against a $(t, q_C, q_E)$ finding attributes adversary $\mathcal{A}_5$ if $\text{Adv}_{\mathcal{A}_5}$ is negligible after making at most $q_C$ RegisterAgent or IssueAttribute queries and $q_E$ Encryption queries in time $t$.

**Additional security properties**

- **Obliviousness**: [LDB03a] The sender cannot tell whether the receiver possesses the policy attributes at the end of each interaction.

- **Forward Repudiability**: [BDS+03, CJT04] Suppose an honest agent member $U_1$ keeps some transcripts of records of interactions between himself and another honest agent member $U_2$. $U_1$ cannot convince any other third party $U_3$ (which is an honest agent member too) that he has interacted with $U_2$, even by revealing its own private key, credentials, and showing $U_3$ all the transcripts between himself and $U_2$. The forwarding repudiability in our scheme is supported by the following facts. If $U_1$ and $U_2$ can communicate successfully, then the exactly same copies of the transcripts between $U_1$ and $U_2$ can be generated by either of them. Therefore, $U_1$ always has enough information to generate the transcripts of records between $U_1$ and $U_2$. Therefore, these transcripts of records could be completely forged by $U_1$, and cannot be used to convince any third party that they were generated by $U_2$.

- **Unlinkability**: No adversary is able to associate receivers of two encrypted messages, even if the adversary is a valid receiver of these two encrypted messages.

### 5.1.4 Complexity Assumptions

We recall the complexity definitions and the assumptions which will be used in our security proofs.
5.2. The Proposed Scheme

**Definition 5.6 (Bilinear Diffie-Hellman (BDH) Problem.)** Given cyclic groups \( G_1, G_2 \) of the order \( q \) together with a bilinear pairing \( \hat{e} : G_1 \times G_1 \to G_2 \), \( g \) is a generator of group \( G_1 \), the BDH problem is \((t, \epsilon)\)-hard if for all \( t \)-time adversary \( A \) we have

\[
\text{Adv}^{\text{BDH}}_A = |\Pr[A(g, g^a, g^b, g^c) = \hat{e}(g, g)^{abc}]| < \epsilon.
\]

**Definition 5.7 (Bilinear Diffie-Hellman (BDH) Assumption.)** There exists no polynomial time algorithm which can solve the BDH problem.

**Definition 5.8 (Decisional Bilinear Diffie-Hellman (DBDH) Problem.)** Given cyclic groups \( G_1, G_2 \) of the order \( q \) together with a bilinear pairing \( \hat{e} : G_1 \times G_1 \to G_2 \), \( g \) is a generator of group \( G_1 \), the DBDH problem is to distinguish between tuples of \((g, g^a, g^b, g^c, \hat{e}(g, g)^{abc})\) and \((g, g^a, g^b, g^c, \tau)\) where \( a, b, c \leftarrow \mathbb{Z}_q^* \) and \( \tau \leftarrow G_2 \). The DBDH problem is \((t, \epsilon)\)-hard if for all \( t \)-time adversaries \( A \) we have

\[
\text{Adv}^{\text{DBDH}}_A = |\Pr[A(g, g^a, g^b, g^c, \hat{e}(g, g)^{abc}) = 1] - \Pr[A(g, g^a, g^b, g^c, \tau) = 1]| < \epsilon.
\]

**Definition 5.9 (Decisional Bilinear Diffie-Hellman (DBDH) Assumption.)** We assume that the probability of a polynomial time algorithm to solve the DBDH problem is negligible.

### 5.2 The Proposed Scheme

In this section, we formally present our scheme.

**Setup.** Let \((G_1, G_2)\) be the bilinear groups defined in Chapter 2. Let \( g_1 \) be the generator of \( G_1 \). The group manager (GM) selects a random \( s \leftarrow \mathbb{Z}_q^* \) as the system master key and sets the system master public key \( pk = g_1^s \). Let \( H : \{0, 1\}^* \to G_1 \), \( H_2 : G_2 \to \{0, 1\}^n \), \( H_3 : \{0, 1\}^n \times \{0, 1\}^n \to \mathbb{Z}_q^* \), \( H_4 : \{0, 1\}^n \to \{0, 1\}^n \), \( H_5 : G_2 \to \mathbb{Z}_q^* \) be five secure cryptographic hash functions. GM maintains a group member list \( L_m \) and an attribute list \( L_a \), which store the pseudonyms of the group members and the certified attributes respectively. \( L_m \) and \( L_a \) are initially empty. The system parameters are:

\[
\text{Params} = (q, G_1, G_2, n, \hat{e}, g_1, pk, H, H_2, H_3, H_4, H_5, L_m, L_a).
\]

**RegisterAgent.** For an agent with identity \( \text{ID}_i \), GM computes the pseudonym \( h_i = H(\text{ID}_i) \) and the private key \( d_i = h_i^s \). GM registers this agent by setting \( L_m := L_m \cup \{h_i\} \) if \( h_i \notin L_m \) and returns the pair \((d_i, h_i)\) to the agent \( \text{ID}_i \).
5.2. The Proposed Scheme

**RevokeAgent.** GM revokes an agent whose pseudonym is $h_i$ by setting $\mathcal{L}_m := \mathcal{L}_m \backslash \{h_i\}$.

**IssueAttribute.** GM issues the attribute $\text{Atb}_i$ to a group member by issuing the credential $\text{Cred}_i = H(\text{Atb}_i)^{\ast}$. GM sets $\mathcal{L}_a := \mathcal{L}_a \cup \{\text{Atb}_i\}$ if $\text{Atb}_i \notin \mathcal{L}_a$.

**SendMsg.** Suppose an agent whose pseudonym is $h_i$ wants to send a message $M$ encrypted with policy

$$POL = \bigvee_{j=1}^{l} \left[ h_j^{\ast} \bigwedge_{k=1}^{l_j} \text{Atb}_k^{[j]} \right].$$

Do the following:

1. Select $z \leftarrow \{0,1\}^n$, $\mu \leftarrow \mathbb{Z}_q^\ast$, and set $r = H_3(z, M)$.

2. Set the ciphertext to be:

$$C = \{g_1^\mu, h_i^r, A, M \oplus H_4(z), \text{ATB-SET}\},$$

where

$$A = z \oplus H_2(\hat{e}(d_i, h_i^{\ast}))^r + \left\{ \bigoplus_{k=1}^{l_j} H_2(\hat{e}(d_i, H(\text{Atb}_k^{[j]}))^r) \right\},$$

$$d_i = h_i^{\ast}, \text{ ATB-SET} = \left\{ l_j + 1, H_5(\hat{e}(h_j^{\ast} \prod_{k=1}^{l_j} H(\text{Atb}_k^{[j]}), pk)^\mu) \right\}, 1 \leq j \leq l.$$


**RcvMsg.** Suppose an agent whose pseudonym is $h_\theta^\ast$ possesses the credentials which satisfy the sender’s policy $POL$. We assume that the agent satisfies one CNF in $POL$ which is

$$h_\theta^{\ast} \bigwedge_{k=1}^{l_\theta} \text{Atb}_k^{[\theta]}.$$ 

The agent decrypts the ciphertext $C$ as follows:

1. Let $C = (X, U, V, W, \text{ATB-SET})$. If $X, U \notin \mathbb{G}_1$, reject the ciphertext.

2. Use the private key to check whether all credential match the sender’s policy attributes as follows:
5.3 Security Analysis

In this section, we present the proofs of the security properties defined in Section 5.1.3:
5.3. Security Analysis

- Adaptive chosen ciphertext secure,
- Impersonation resistance,
- Probing resistance,
- Indistinguishability to eavesdroppers, and
- Hidden credentials.

To prove the adaptive chosen ciphertext security of our scheme, we present the proof in three steps. We first show an adaptive chosen ciphertext attack on our scheme can be converted to an IND-CCA attack on the $\text{BasicPub}^{hy}$ scheme which is from [BF01]. We then use the result of Fujisaki and Okamoto (Theorem 14 in [FO99]), which shows that an IND-CCA attack on $\text{BasicPub}^{hy}$ scheme can be converted to an IND-CPA attack on the $\text{BasicPub}$ scheme [BF01]. We finally make use of the result of Boneh and Franklin (Lemma 4.3 in [BF01]), which shows that an IND-CPA attack on the $\text{BasicPub}$ scheme can be reduced to the BDH problem. Accordingly, we first prove the following Lemma.

**Lemma 5.1** If there exists an adaptive chosen ciphertext adversary $\mathcal{A}$ who has advantage $\epsilon$ against our scheme after making at most $q_H$ queries to the $H$ oracle. Then there exists an IND-CCA adversary $\mathcal{B}$ who has advantage

$$\text{Adv}_{\mathcal{B},\text{IND-CCA}} \geq \frac{\epsilon}{q_H(q_H - 1)^2}$$

against $\text{BasicPub}^{hy}$ with running time $O(\text{time}(\mathcal{A}))$.

**Proof:** An IND-CCA adversary $\mathcal{B}$ uses $\mathcal{A}$ to attack on $\text{BasicPub}^{hy}$. The game between the challenger $\mathcal{C}$ and the adversary $\mathcal{B}$ starts with the challenger first generating random system parameters: $(q, \mathbb{G}_1, \mathbb{G}_2, e, n, g_1, pk, Q_{\text{ID}}, H_2, H_3, H_4)$ and a private key $d_{\text{ID}} = Q^{*}_{\text{ID}}$. The challenger gives all the system parameters except private key to adversary $\mathcal{B}$. $\mathcal{B}$ then simulates the challenger and interacts with $\mathcal{A}$ as follows:

**Setup:** $\mathcal{B}$ gives $\mathcal{A}$ the system parameters: $(q, \mathbb{G}_1, \mathbb{G}_2, e, n, g_1, pk, H, H_2, H_3, H_4, H_5, \mathcal{L}_m, \mathcal{L}_a)$. $H, H_5$ are the random oracles controlled by $\mathcal{B}$ as described below. $\mathcal{L}_m, \mathcal{L}_a$ are the group member list and the attribute list respectively. They are initially empty. $\mathcal{B}$ randomly picks $\delta_1, \delta_2 \leftarrow \{1, 2, \ldots, q_H\}$ and $c \leftarrow \mathbb{Z}_q^*$. Keep $\delta_1, \delta_2, c$ as secrets against adversary $\mathcal{A}$. 
5.3. Security Analysis

**H-queries:** $\mathcal{A}$ can make at most $q_H$ queries to the random oracle $H$ at any time. To respond to these queries, $\mathcal{B}$ maintains an $H$-list of tuples $(\text{Atb}_i/\text{ID}_i, Q_i, b_i, i)$. The list is initially empty. Here $i$ represents the sequence of the queries ($1 \leq i \leq q_H$). For each query $\text{Atb}_i/\text{ID}_i$, $\mathcal{B}$ responds as follows:

1. If $\text{Atb}_i/\text{ID}_i$ appears in a tuple $(\text{Atb}_i/\text{ID}_i, Q_i, b_i, i)$ of the $H$-list, $\mathcal{B}$ responds to $\mathcal{A}$ with $Q_i$.

2. If $i = \delta_1$, compute $Q_i = g_1^i$ and add $(\text{Atb}_i/\text{ID}_i, Q_i, c, \delta_1)$ to the $H$-list; If $i = \delta_2$, compute $Q_i = Q_1^{1/i} \text{ID}_i$ and add $(\text{Atb}_i/\text{ID}_i, Q_i, \frac{1}{c}, \delta_2)$ to the $H$-list. $\mathcal{B}$ responds to $\mathcal{A}$ with $Q_i$.

3. Otherwise, if $i \neq \delta_1$ and $i \neq \delta_2$, pick a random $b \in \mathbb{Z}_q^*$ and compute $Q_i = g_1^b$.

   Add $(\text{Atb}_i/\text{ID}_i, Q_i, b, i)$ to the $H$-list. $\mathcal{B}$ responds to $\mathcal{A}$ with $Q_i$.

**H3-queries:** To respond to these queries, $\mathcal{B}$ maintains a list, $H_3$-list of tuples $(z_i, M_j, r_{ij})$. The list is initially empty. For each query $(z_i, M_j)$, $\mathcal{B}$ responds as follows:

1. If $(z_i, M_j)$ appears in a tuple $(z_i, M_j, r_{ij})$ on the $H_3$-list, $\mathcal{B}$ responds to $\mathcal{A}$ with $r_{ij}$.

2. Otherwise, $\mathcal{B}$ submits $(z_i, M_j)$ to the challenger $\mathcal{C}$ (the $H_3$ random oracle). In response, $\mathcal{C}$ returns $r$ to $\mathcal{B}$. $\mathcal{B}$ adds $(z_i, M_j, r)$ to the $H_3$-list and responds to $\mathcal{A}$ with $r$.

**H4-queries:** To respond to these queries, $\mathcal{B}$ maintains a list, $H_4$-list of tuples $(z_i, h_{4i})$. The list is initially empty. For each query $z_i$, $\mathcal{B}$ responds as follows:

1. If $z_i$ appears in a tuple $(z_i, h_{4i})$ of the $H_4$-list, $\mathcal{B}$ responds to $\mathcal{A}$ with $h_{4i}$.

2. Otherwise, $\mathcal{B}$ submits $z_i$ to the challenger $\mathcal{C}$ (the $H_4$ random oracle). In response, $\mathcal{C}$ returns $h$ to $\mathcal{B}$. $\mathcal{B}$ adds $(z_i, h)$ to the $H_4$-list and responds to $\mathcal{A}$ with $h$.

**Phase 1: RegisterAgent/IssueAttribute queries.** For each query $\text{ID}_i/\text{Atb}_i$ issued by adversary $\mathcal{A}$, $\mathcal{B}$ responds to this query as follows:

1. Run the above $H$-queries algorithm to obtain a $Q_i \in G_1$ such that $H(\text{Atb}_i)/H(\text{ID}_i) = Q_i$. Assume $(\text{Atb}_i/\text{ID}_i, Q_i, b_i, i)$ is the corresponding tuple on the $H$-list. If $i = \delta_2$, then $\mathcal{B}$ reports failure and terminates.
2. Otherwise, \( i \neq \delta_2 \); hence \( Q_i = g_{1_i}^{h_i} \). The corresponding credential/private key is 
\[ C_{\text{red},i} / d_i = Q_i^* = (g_{1_i}^{h_i})^* = (g_{1_i}^{h_i}) = pk_i. \]
If the query is a RegisterAgent query, \( B \) returns \((Q_i, d_i)\) to \( A \) and sets \( L_m := L_m \cup \{Q_i\} \) if \( Q_i \notin L_m \). If the query is an IssueAttribute query, \( B \) returns \( C_{\text{red},i} \) to \( A \) and sets \( L_a := L_a \cup \{Atb_i\} \) if \( Atb_i \notin L_a \).

**Phase 1: Decryption queries.** For each query \((C_i, \mathcal{POL}_j)\) issued by adversary \( A \), let \( \mathcal{POL}_j = h_j \bigwedge_{k=1}^{l_j} \text{Atb}_{jk} \) (\( h_j \) is the pseudonym of an agent and \( \text{Atb}_{jk} \) is the attribute). \( B \) responds to this query as follows:

1. Run the above \( H \)-queries algorithm to obtain \( Q_{jk} \) for all the \( \text{Atb}_{jk} \) \( (1 \leq k \leq l_j) \). 
If none of the \( h_j, Q_{j1}, ..., Q_{jl_j} \) equals \( Q_{\text{id}} \), then \( B \) can run the above RegisterAgent/IssueAttribute queries algorithm to obtain the private key \( d_j \) of \( h_j \) and the credentials \( C_{\text{red},jk} \) of \( \text{Atb}_{jk} \) for all the \( k \) \( (1 \leq k \leq l_j) \). \( B \) then can do the decryption and give the answer to \( A \) by himself.

2. Otherwise, if one of the \( h_j, Q_{j1}, ..., Q_{jl_j} \) equals \( Q_{\text{id}} \), conduct the following. Let \( C_i = (X_i, U_i, V_i, W_i, \text{ATB-SET}_i) \). \( C_i \) is a valid ciphertext, if and only if for some unknown \( r_i, z_i, M_i, h_i \) satisfy: \( U_i = h_i^{r_i}, W_i = M_i \oplus H_4(z_i), r_i = H_3(z_i, M_i) \) and \( h_i \in L_m \). Here, \( h_i \) is the sender’s pseudonym and \( M_i \) is the plaintext of \( C_i \). Therefore, there must exist tuples \((z_i, M_i, r_i), (z_i, h_i)\) for \( r_i, z_i, M_i, h_i \) on the \( H_3\)-list, \( H_4\)-list respectively. Otherwise, \( A \) has negligible probability to make a decryption query with a valid \( C_i \). \( B \) searches the \( H_3\)-list, \( H_4\)-list simultaneously for such a \( z^* \) that appears on both lists. If tuples \((z^*, M^*, r^*), (z^*, h_i^*)\) appear on the \( H_3\)-list, \( H_4\)-list respectively, \( B \) checks whether \( W_i^* \doteq M^* \oplus h_i^* \). Then, \( B \) can use \( r^* \) to check the validity of \( U_i \) and \( V_i \) as follows: \( B \) first computes \( h_i = U_i^{1/r^*} \) as the sender’s pseudonym. Then, \( B \) can compute the sender’s private key \( d_i = (g_{1_i}^{h_i})^* = pk_i \). Suppose \((\text{ID}_i, h_i, b_i, i)\) is the tuple corresponding to \( h_i \) on the \( H\)-list (Recall that one of the \( h_j, Q_{j1}, ..., Q_{jl_j} \) equals \( Q_{\text{id}} \), so \( h_i \neq Q_{\text{id}} \)). Hence the private key of \( h_i \) can be computed. With the help of \( d_i, z^*, r^* \), the validity of \( V_i \) can be easily found. If \( V_i \) is not valid, \( B \) chooses another appropriate item from \( H_3\)-list, \( H_4\)-list to conduct the same computation again. Finally, \( B \) can output a valid pair \((M_i, h_i)\), where \( M_i \) is the plaintext of \( C_i \) and \( h_i \) is the sender’s pseudonym. \( B \) responds to \( A \) with \((M_i, h_i)\).
5.3. Security Analysis

Challenge: Adversary A outputs a pair ($\mathcal{POL}, h_S$) and two messages $M_0, M_1$ on which he wishes to be challenged. Here $h_S$ is the sender’s pseudonym. Suppose $\mathcal{POL} = h_\beta \land_{k=\beta_1}^\beta \mathcal{At}_b$. B responds as follows:

1. B submits $M_0, M_1$ to C as the messages on which he wishes to be challenged. C responds with a BasicPub ciphertext $C = (U, V, W)$ such that $C$ is the encryption of $M_\omega$ for a random $\omega \in \{0, 1\}$. Let $U = g_1^r$ for some $r$ unknown to B. Run the above $H$-queries algorithm to obtain $Q_k$ such that $H(\text{At}_b) = Q_k$ for all the $k$ ($\beta_1 \leq k \leq \beta_t$). Assume $(\text{ID}_\beta, h_\beta, b_\beta, \beta)$, $(\text{At}_b, Q_k, b_k, k)$ ($\beta_1 \leq k \leq \beta_t$) are the corresponding tuples of $h_\beta$, $\text{At}_b$ ($\beta_1 \leq k \leq \beta_t$) respectively on the $H$-list.

2. If one of $\beta, \beta_1, ..., \beta_t$ equals $\delta_2$ and $h_S = g_1^c$, suppose $\beta = \delta_2$ (the same for the other situation, so we omit it here). Therefore, $h_\beta = Q_1^c$. Let

$$V = z \oplus H_2(\hat{e}(pk, Q_{\text{ID}})^r) = z \oplus H_2(\hat{e}((g_1^c)^{c}, Q_{\text{ID}}^1)^r)$$

$$= z \oplus H_2(\hat{e}((g_1^c)^s, Q_{\text{ID}}^1)^r) = z \oplus H_2(\hat{e}((g_1^c)^s, h_\beta)^r).$$

Set

$$V' = V \oplus \left\{ \bigoplus_{k=\beta_1}^{\beta_t} H_2(\hat{e}(U^c, pk^b)) \right\}$$

$$= z \oplus H_2(\hat{e}((g_1^c)^s, h_\beta)^r) \oplus \left\{ \bigoplus_{k=\beta_1}^{\beta_t} H_2(\hat{e}((g_1^c)^s, (g_k^b)^s)) \right\}$$

$$= z \oplus H_2(\hat{e}((g_1^c)^s, h_\beta)^r) \oplus \left\{ \bigoplus_{k=\beta_1}^{\beta_t} H_2(\hat{e}((g_1^c)^s, Q_k)^r) \right\}.$$ 

Pick a random $\mu_0 \leftarrow \mathbb{Z}_q^*$ and set $X_0 = g_1^{\mu_0}$. Compute

$$\text{ATB-SET} = \left\{ l + 1, H_5(\hat{e}(h_\beta \cdot \prod_{k=\beta_1}^{\beta_t} Q_k, pk)^{\mu_0}) \right\} \text{ and } U' = U^c = (g_1^c)^c = (g_1^c)^r.$$ 

Set $C' = (X_0, U', V', W, \text{ATB-SET})$. From the above equations, it can be seen that $C'$ is a valid encryption of $M_\omega$ under $(h_S, \mathcal{POL})$, if and only if $h_S = g_1^c$.

B responds to A with $C'$ as the challenge.

3. Otherwise, B reports failure and terminates.
Phase 2: RegisterAgent/IssueAttribute queries. $B$ responds to these queries in the same way in Phase 1.

Phase 2: Decryption queries. $B$ responds to these queries in the same way in Phase 1. However, if the decryption query relayed to $C$ is equal to the challenge ciphertext $C = (U, V, W)$, $B$ reports a failure and terminates.

Guess: Eventually, adversary $A$ outputs a guess $\omega'$ for $\omega$. $B$ outputs $\omega'$ as its guess for $\omega$.

$B$ can successfully break the BasicPub$^{by}$ scheme, if and only if:

- $B$ does not abort during the RegisterAgent/IssueAttribute queries in Phases 1 and 2. The probability is at least $(1 - \frac{1}{q_H})^{q_C} \geq \frac{1}{e}$. We assume adversary $A$ made totally $q_C$ RegisterAgent/IssueAttribute queries in Phases 1 and 2.

- $A$’s challenge does not cause $B$ to abort. The probability is at least $1/q_H(q_H - 1)$.

- $B$ does not abort during the Decryption queries in Phase 2. If the decryption query of $A$ relayed to $C$ is equal to the challenge ciphertext $C = (U, V, W)$, then $B$ could not find appropriate items in both of $H_3$-list or $H_4$-list, and hence abort. Suppose $A$ makes $q_{H_4}$ queries to the $H_3$ oracle, $q_{H_4}$ queries to the $H_4$ oracle, $q_{D_2}$ Decryption queries in Phase 2. Therefore, the probability that $B$ does not abort during the Decryption queries in Phase 2 is at least

$$(1 - \frac{1}{(q_{H_3} + 1)(q_{H_4} + 1)})^{q_{D_2}} \geq \frac{1}{e},$$

since $(q_{H_3} + 1)(q_{H_4} + 1)) \geq q_{D_2} + 1$.

- $A$ outputs a correct guess, which has the probability of at least $\epsilon$.

Combining them together, $B$ has advantage

$$Adv_{B,IND-CCA} \geq \frac{\epsilon}{q_H(q_H - 1)e^2}$$

against BasicPub$^{by}$ with running time $O(time(A))$.

To complete the proof of adaptive chosen ciphertext security of our scheme, we need the following results:
Lemma 5.2 (Fujisaki-Okamoto)[FO99] Suppose $A$ is an IND-CCA adversary that achieves advantage $\epsilon$ when attack BasicPub. Suppose $A$ has running time $t$, makes at most $q_D$ decryption queries, and makes at most $q_{H_3}, q_{H_4}$ queries to the hash functions $H_3, H_4$ respectively. Then there is an IND-CPA adversary $B$ against BasicPub with running time $t_1$ and advantage $\epsilon_1$ where
\[
\epsilon_1 \geq F_{\text{adv}}(\epsilon, q_{H_3}, q_{H_4}, q_D) = \frac{1}{2(q_{H_4} + q_{H_3})}[(\epsilon + 1)(1 - 2/q)^q - 1]
\]
\[
t_1 \leq F_{\text{time}}(t, q_{H_4}, q_{H_3}) = t + O((q_{H_4} + q_{H_3}) \cdot n), \text{and}
\]
Here $q$ is the size of the group $\mathbb{G}_1, \mathbb{G}_2$ and $n$ is the length of $\sigma$.

Lemma 5.3 (Boneh-Franklin) [BF01] Let $H_2$ be a random oracle from $\mathbb{G}_2$ to $\{0,1\}^n$. Let $A$ be an IND-CPA adversary that has advantage $\epsilon$ against BasicPub. Suppose $A$ makes a total of $q_{H_2} > 0$ queries to $H_2$. Then there is an algorithm $B$ that solves the BDH problem with advantage at least $2\epsilon/q_{H_2}$ and a running time $O(\text{time}(A))$.

Now, we present the proof of adaptive chosen ciphertext security of our scheme.

Theorem 5.4 Our scheme is adaptive chosen ciphertext secure under the BDH assumption in $(\mathbb{G}_1, \mathbb{G}_2, \hat{e})$. Concretely, if there is an adversary $A$ has advantage $\epsilon$ against our scheme with running time $t$, suppose $A$ makes at most $q_D$ Decryption queries, and at most $q_H, q_{H_2}, q_{H_3}, q_{H_4}$ queries to the hash functions $H, H_2, H_3, H_4$ respectively, then there exists an algorithm $B$ which can solve the BDH problem with running time $t_1$ where:
\[
\text{Adv}_B \geq 2F_{\text{adv}}(\epsilon, q_H(q_H - 1)e^2, q_{H_1}, q_{H_3}, q_D)/q_{H_2},
\]
\[
t_1 \leq F_{\text{time}}(t, q_{H_4}, q_{H_3}),
\]
where the function $F_{\text{adv}}$ and $F_{\text{time}}$ are defined in Lemma 5.2.

Proof: Lemma 5.1 shows that an adversary $A$ in our scheme implies an IND-CCA adversary on BasicPub. Lemma 5.2 shows that an IND-CCA adversary on BasicPub implies an IND-CPA adversary on BasicPub. Lemma 5.3 shows that an IND-CPA adversary on BasicPub implies an algorithm for BDH. Composing all these reductions gives the required bounds.

Now, we present the proof of the impersonation resistance property of our scheme.
Theorem 5.5  Our scheme is impersonation resistant under the BDH assumption in \((G_1, G_2, \hat{e})\). Concretely, if there is an adversary \(A\) that has advantage \(\epsilon\) against the impersonation resistance of our scheme; Suppose \(A\) makes at most \(q_E\) Encryption queries, at most \(q_H\) queries to the \(H\) oracle, a total \(q_{H_2} > 0, q_{H_3} > 0\) queries to the \(H_2, H_3\) oracles respectively, then there exists an algorithm \(B\) which can solve the BDH problem with advantage

\[
\text{Adv}_B \geq (1 - \frac{1}{q_H})^{q_E} \frac{\epsilon}{q_{H_2} q_{H_3}}
\]

in running time \(O(\text{time}(A))\).

Proof: Assume that an adversary \(A\) violates the impersonation resistance property. We now show how to construct an algorithm \(B\) which can solve the BDH problem. Suppose \(B\) is given \((q, G_1, G_2, \hat{e}, g, g^a, g^b, g^c)\) as an instance of the BDH problem. Let \(D = \hat{e}(g, g)^{abc}\) be the solution to this BDH problem. Algorithm \(B\) interacts with \(A\) as follows:

**Phase 1:** \(B\) gives the system parameters \((q, G_1, G_2, \hat{e}, n, g, pk, H, H_2, H_3, H_4, H_5, L_m, L_a)\) to \(A\) by setting \(pk = g^a\). Here \(H, H_2, H_3\) are the random oracles controlled by \(B\) as described below.

**Phase 2:** \(A\) chooses the target sender \(ID_S\) and receiver \(ID_R\). \(B\) sets \(h_S = g^b, h_R = g^c\) as the pseudonyms for \(ID_S, ID_R\) respectively. Add \((ID_S, h_S, -), (ID_R, h_R, -)\) to the \(H\)-list which is described below. Set \(L_m := L_m \cup \{h_S\}, L_m := L_m \cup \{h_R\}\).

**Phase 3:** \(H\)-queries. To respond to each query, \(B\) maintains a list \(H\)-list of tuples \((Atb_i, ID_i, Q_i, x_i)\). The list is initially empty. For a query \(Atb_i, ID_i \in \{0, 1\}^*\), \(B\) responds as follows:

1. If \(Atb_i, ID_i\) already exists on the list in a tuple \((Atb_i, ID_i, Q_i, x_i)\), \(B\) responds to \(A\) with \(Q_i\).

2. Otherwise, \(B\) picks a random \(x \leftarrow \mathbb{Z}_q^*\) and computes \(Q_i = g^x\). Add \((Atb_i, ID_i, Q_i, x)\) to the \(H\)-list. \(B\) responds to \(A\) with \(Q_i\).

**Phase 3:** \(H_2\)-queries. To respond to each query, \(B\) maintains a list \(H_2\)-list of tuples \((y_{2i}, h_{2i})\). The list is initially empty. For a query \(y_{2i} \in G_2\), \(B\) responds as follows:

1. If \(y_{2i}\) is already on the \(H_2\)-list in a tuple \((y_{2i}, h_{2i})\), then respond with \(H_2(y_{2i}) = h_{2i}\).
2. Otherwise, $B$ picks a random string $h_{2i} \in \{0, 1\}^n$ and adds the tuple $(y_{2i}, h_{2i})$ to the $H_2$-list. $B$ responds to $A$ with $H_2(y_{2i}) = h_{2i}$.

**Phase 3: $H_3$-queries.** To respond to each query, $B$ maintains a list $H_3$-list of tuples $(z_{3i}, M_{3i}, h_{3i})$. The list is initially empty. For a query $(z_{3i}, M_{3i}) \in \{0, 1\}^n \times \{0, 1\}^n$, $B$ responds as follows:

1. If $(z_{3i}, M_{3i})$ is already on the $H_2$-list in a tuple $(z_{3i}, M_{3i}, h_{3i})$, then respond with $H_3(z_{3i}, M_{3i}) = h_{3i}$.

2. Otherwise, $B$ picks a random $h_{3i} \leftarrow \mathbb{Z}_q^*$ and adds the tuple $(z_{3i}, M_{3i}, h_{3i})$ to the $H_3$-list. $B$ responds to $A$ with $H_3(z_{3i}, M_{3i}) = h_{3i}$.

**Phase 3: RegisterAgent/IssueAttribute queries.** Suppose $Atb_i/ID_i$ is a query issued by adversary $A$. $B$ responds to this query as follows:

1. Run the above $H$-queries algorithm to obtain a $Q_i \in G_1$ such that $H(Atb_i)/H(ID_i) = Q_i$. Assume $(Atb_i/ID_i, Q_i, x_i)$ is the corresponding tuple on the $H$-list.

2. $B$ computes $\text{Cred}_i/d_i = Q^a_i = (g^{x_i})^a = (g^a)^{x_i}$. If the query is a RegisterAgent query, $B$ returns $(Q_i, d_i)$ to $A$ and sets $\mathcal{L}_m := \mathcal{L}_m \cup \{Q_i\}$ if $Q_i \notin \mathcal{L}_m$; If the query is an IssueAttribute query, $B$ returns $\text{Cred}_i$ to $A$ and sets $\mathcal{L}_a := \mathcal{L}_a \cup \{Atb_i\}$ if $Atb_i \notin \mathcal{L}_a$.

**Phase 3: Encryption queries.** For a query $(h_i, POL_i, M_i)$ issued by $A$ where $POL_i = \bigvee_{j=1}^{h_i} POL_{ij}$. $POL_{ij}$ is the concatenation of one pseudonym and any number of attributes. $B$ responds as follows:

1. If $h_i = h_R$ and $h_S \in POL_i$, $B$ reports failure and terminates.

2. Otherwise, $B$ can run the RegisterAgent/IssueAttribute queries algorithm to obtain either the private key $d_i$ of $h_i$, or all the corresponding credentials of $POL_i$. Either way $B$ is able to encrypt message $M_i$ and responds to $A$ by itself.

**Phase 4:** $A$ outputs a ciphertext $C$. At this point, $B$ picks two random tuples $(y_{2j}, h_{2j})$, $(z_{3k}, M_{3k}, h_{3k})$ from the $H_2$-list, $H_3$-list respectively and outputs $y_{2j}^{1/h_{3k}}$ as the solution to the given instance of the BDH problem.
5.3. Security Analysis

Analysis: If $\mathcal{A}$ can output a valid $C$ in the above game, then $\mathcal{A}$ must have issued a query for $H_2(D^r)$ at some point during the simulation above (this implies that at the end of the simulation $D^r$ appears in some tuple on $H_2$-list). Let $C = (X, U, V, W, ATB-SET)$ and $U = (h_S)^r$ for some unknown $r$ to $\mathcal{B}$. Denote $d_S$ as the private key of $h_S$. To observe this fact we assume $C$ is encrypted under the policy $\mathcal{POL} = h_R \wedge_{k=\alpha_1}^{\alpha_l} (Atb_k)$ for some $Atb_k$ where $\alpha_a \leq k \leq \alpha_l$. So

$$V = z \oplus H_2(\hat{e}(d_S, h_R)^r) \oplus \left\{ \bigoplus_{k=\alpha_1}^{\alpha_l} H_2(\hat{e}(d_S, H(Atb_k))^r) \right\}$$

$$= z \oplus H_2(\hat{e}((g^b)^a, g^c)^r) \oplus \left\{ \bigoplus_{k=\alpha_1}^{\alpha_l} H_2(\hat{e}(d_S, H(Atb_k))^r) \right\}$$

$$= z \oplus H_2(D^r) \oplus \left\{ \bigoplus_{k=\alpha_1}^{\alpha_l} H_2(\hat{e}(d_S, H(Atb_k))^r) \right\}.$$ 

If $\mathcal{A}$ never issues a query for $H_2(D^r)$, then $C$ is independent of $\mathcal{A}$’s view (since $V$ is independent of $\mathcal{A}$’s view). Therefore, $\mathcal{A}$’s advantage against impersonation resistance is equal to zero. But by definition $\mathcal{A}$ has advantage of at least $\epsilon$. So $\mathcal{A}$ must have queried $H_2(D^r)$ at some point during the execution of the game above.

On the other hand, for the same reason, if $C$ is valid, then $\mathcal{A}$ must have queried $H_3$ oracle for such a value $r$ since $r = H_3(z, M)$. Otherwise, $r$ is independent of $\mathcal{A}$’s view. So there must exist such an entry $(z_m, M_m, r)$ on the $H_3$-list. The probability of $\mathcal{B}$ solving the given instance of the BDH problem depends on the following events:

- $B$ does not abort during the encryption queries. The probability is

$$\left(1 - \frac{l_i}{q_H(q_H - 1)}\right)^{q_E} \geq \left(1 - \frac{1}{q_H}\right)^{q_E}.$$ 

We recall that $POL_i = \bigvee_{j=1}^{l_i} POL_{ij}$ in each encryption query.

- $A$ successfully outputs a valid $C$. The probability is at least $\epsilon$.

- $B$ outputs the correct answer $D$. The probability is $\frac{1}{q_{H_2}q_{H_3}}$.

Therefore, the probability of $\mathcal{B}$ solving the given instance of the BDH problem is

$$\text{Adv}_B \geq \left(1 - \frac{1}{q_H}\right)^{q_E} \frac{\epsilon}{q_{H_2}q_{H_3}}.$$ 

Now, we present the proof of the probing resistance property of our scheme.
5.3. Security Analysis

Theorem 5.6 Our scheme is probing resistant under the DBDH assumption in $(\mathbb{G}_1, \mathbb{G}_2, \hat{e})$. Concretely, if there is an adversary $A$ that has advantage $\epsilon$ against the probing resistance of our scheme; Suppose $A$ makes at most $q_H$ queries to the $H$ oracle, at most $q_E$ Encryption queries, then there exists an algorithm $B$ which can solve the DBDH problem with advantage

$$Adv_B \geq (1 - \frac{1}{q_H}) q_E \frac{\epsilon}{e}$$

in running time $O(\text{time}(A))$.

Proof: We construct an algorithm $B$ which uses $A$’s advantage to solve the DBDH problem. Suppose $B$ is given $(q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, g, g^a, g^b, g^c, \tau)$ as an instance of the DBDH problem. Algorithm $B$ interacts with $A$ as follows:

Phase 1: As in Theorem 5.5.

Phase 2: $A$ chooses the target sender $ID_S$, a policy $POL$ where $POL = h_A \bigwedge_{k=\alpha_1}^\alpha$ (Atb$_k$) and a message $M \in \{0, 1\}^n$. $B$ sets $h_S = g^b$ as the pseudonyms for $ID_S$. Pick a random $\theta \overset{\$}{\leftarrow} \mathbb{Z}_q^*$ and set $h_A = g^\theta$ as the pseudonym for $A$. Pick a random attribute Atb$_\beta \overset{\$}{\leftarrow} POL$. Set $\text{Cred}_\beta = H(\text{Atb}_\beta) = g^c$. Add $(ID_A, h_A, \theta), (ID_S, h_S, -), (\text{Atb}_\beta, \text{Cred}_\beta, -)$ to the $H$-list which is described below. Set $L_m := L_m \cup \{h_S\}, L_m := L_m \cup \{h_A\}$ and $L_a := L_a \cup \{\text{Atb}_\beta\}$.

Phase 3: $H$-queries. As in Theorem 5.5.

Phase 3: RegisterAgent/IssueAttribute queries. As in Theorem 5.5 except if the query is Atb$_\beta$, $B$ reports failure and terminates.

Phase 3: Encryption queries. For a query $(h_i, POL_i, M_i)$ issued by $A$ where $POL_i = \bigvee_{j=1}^{l_i} (POL_{ij})$, where $POL_{ij}$ is the concatenation of one pseudonym and any number of attributes, $B$ responds as follows:

1. If $h_i = h_S$ and Atb$_\beta \in POL_i$, $B$ reports failure and terminates.

2. Otherwise, $B$ can run the RegisterAgent/IssueAttribute queries algorithm to obtain either the private key $d_i$ of $h_i$ or all the corresponding credentials of $POL_i$. In either way, $B$ is able to encrypt message $M_i$ and responds to $A$.

Phase 4: $B$ tosses a coin $\omega \leftarrow \{0, 1\}$.

Phase 5: $B$ picks a random $\mu \overset{\$}{\leftarrow} \mathbb{Z}_q^*$ and sets $X = g^\mu$. Compute

$$\text{ATB-SET} = \left\{ l + 1, H_5(\hat{e}(h_A \cdot \prod_{k=\alpha_1}^{\alpha_i} H(\text{Atb}_k), pk)^\mu) \right\}.$$
Pick a random $z \leftarrow \{0,1\}^n$, compute $r = H_3(z, M)$. Set $U = (h_S)^r = (g^b)^r$. Compute

$$V = z \oplus H_2(\hat{e}((h_S)^a, h_A)^r) \oplus \left\{ \bigoplus_{k=\alpha_1}^{\alpha_l} H_2(\hat{e}((h_S)^a, H(Atb_k))^{r}) \right\}$$

$$= z \oplus H_2(\hat{e}(h_S, h_A)^r) \oplus H_2(\hat{e}(h_S, H(Atb_k)^a)^r) \oplus \left\{ \bigoplus_{k=\alpha_1}^{\alpha_l} H_2(\hat{e}(h_S, H(Atb_k)^a)^r) \right\}$$

$$= z \oplus H_2(\hat{e}(g^b, (g^a)^\theta)^r) \oplus H_2(\tau^r) \oplus \left\{ \bigoplus_{k=\alpha_1}^{\alpha_l} H_2(\hat{e}(g^b, (g^a)^x_k)^r) \right\}$$

and $W = M \oplus H_4(z)$. Set $C = (X, U, V, W, ATB-SET)$ and responds to $\mathcal{A}$ with $C$. Here we assume the tuples $(Atb_k, Q, x_k)$ (for all the $k, \alpha_1 \leq k \leq \alpha_l$) already exist on the $H$-list. Otherwise, $\mathcal{B}$ can make queries to obtain these.

**Phase 6:** $\mathcal{A}$ outputs its guess $\omega' \in \{0, 1\}$. If $\omega' = 0$, $\mathcal{B}$ outputs TRUE as the solution for the given instance of the DBDH problem. If $\omega' = 1$, $\mathcal{B}$ outputs FALSE as the solution for the given instance of the DBDH problem.

**Analysis:** Observe the challenge ciphertext $C$ we constructed in the game. Whether $C$ is a valid ciphertext of $(ID_S, POL, M)$, depends on the validity of the part $V$. Observing $V$, we can find that: if $\tau = \hat{e}(g, g)^{abc}$ then $C$ is the valid ciphertext of $(ID_S, POL, M)$; if $\tau \neq \hat{e}(g, g)^{abc}$ then $C$ is equal to a random simulation. Therefore, $\mathcal{B}$ can solve the given instance of the DBDH problem if $\mathcal{A}$ can decide the validity of the ciphertext $C$. The probability of $\mathcal{B}$ solving the given instance of the DBDH problem depends on the following events:

- $\mathcal{B}$ does not abort during the RegisterAgent/IssueAttribute queries. The probability is at least $(1 - 1/q_H)^{ac} \geq 1/e$. We assume adversary $\mathcal{A}$ made at most $q_C$ RegisterAgent/IssueAttribute queries.

- $\mathcal{B}$ does not abort during the Encryption queries. The probability is at least $(1 - 1/q_H)^{GE}$.

- $\mathcal{A}$ outputs a correct guess. The probability is at least $\epsilon$.

Therefore, the probability that $\mathcal{B}$ can solve the given instance of the DBDH problem is

$$Adv_B \geq (1 - \frac{1}{q_H})^{q_E} \frac{\epsilon}{e}.$$
5.3. Security Analysis

In the following, we present the proof of the indistinguishable to eavesdroppers property of our scheme.

**Theorem 5.7** Our scheme is indistinguishable to eavesdroppers under the DBDH assumption in \((G_1, G_2, \hat{e})\). Concretely, if there is an adversary \(A\) which has advantage \(\epsilon\) against indistinguishability to eavesdroppers of our scheme, suppose \(A\) makes at most \(q_H\) queries to the \(H\) oracle, at most \(q_E\) Encryption queries, then there exists an algorithm \(B\) which can solve the DBDH problem with advantage

\[
\text{Adv}_B \geq (1 - \frac{1}{q_H})^{q_E}\epsilon
\]

in running time \(O(\text{time}(A))\).

**Proof:** The proof is similar to the proof of Theorem 5.6. It is therefore omitted.

We now present the proof of the hidden credential property of our scheme.

**Theorem 5.8** Our scheme possesses the property of hidden credential under the DBDH assumption in \((G_1, G_2, \hat{e})\). Concretely, if there is an adversary \(A\) that has advantage \(\epsilon\) against hidden credential of our scheme, then there exists an algorithm \(B\) which can solve the DBDH problem with advantage \(\text{Adv}_B \geq \epsilon/e\) in running time \(O(\text{time}(A))\).

**Proof:** We construct an algorithm \(B\) that use \(A\)’s advantage to solve the DBDH problem. Suppose \(B\) is given \((q, G_1, G_2, \hat{e}, g, g^a, g^b, g^c, \tau)\) as an instance of the DBDH problem. Algorithm \(B\) interacts with \(A\) as follows:

**Phase 1:** As in Theorem 5.5.

**Phase 2:** \(A\) chooses target sender \(\text{ID}_S\), a policy \(POL = h_{\sigma} \land_{k=\alpha_1}^{\alpha_7} (Atb_k)\) and a message \(M \in \{0,1\}^n\). Pick a random attribute \(Atb_\gamma \leftarrow POL\). Set \(\text{Cred}_\gamma = H(Atb_\gamma) = g^c\). Add \((Atb_\gamma, \text{Cred}_\gamma, -)\) to the \(H\)-list which is described below. Set \(L_a := L_a \cup \{Atb_\gamma\}\).

**Phase 3:** \(H\)-queries. As in Theorem 5.5.

**Phase 3:** RegisterAgent/IssueAttribute queries. As in Theorem 5.5 except if the query is \(Atb_\gamma\), \(B\) reports failure and terminates.

**Phase 3:** Encryption queries. For a query \((h_i, POL_i, M_i)\) issued by \(A\) where \(POL_i = \bigvee_{j=1}^{k_i} (POL_{ij})\), where \(POL_{ij}\) is the concatenation of one pseudonym and
any number of attributes, \( \mathcal{B} \) first runs the RegisterAgent/IssueAttribute queries algorithm to obtain the private key \( d_i \) of \( h_i \). Then, \( \mathcal{B} \) runs the algorithm SendMsg using \((d_i, POL_i, M_i)\) and returns the result to \( \mathcal{A} \).

**Phase 4:** \( \mathcal{B} \) tosses a coin \( \omega \leftarrow \{0, 1\} \).

**Phase 5:** \( \mathcal{B} \) sets \( X = g^b \). Compute

\[
ATB\text{-}SET = \begin{cases} 
  l + 1, H_5(\hat{e}(h_\sigma \cdot \prod_{k=\alpha_1}^{\alpha_1, k \neq \gamma} H(Atb_k), \quad pk)^b) \\
  \end{cases}
\]

\[
= \begin{cases} 
  l + 1, H_5(\hat{e}(g^{x_\sigma} \cdot \prod_{k=\alpha_1}^{\alpha_1, k \neq \gamma} H(Atb_k), \quad pk)^b) \\
  \end{cases}
\]

\[
= \begin{cases} 
  l + 1, H_5(\hat{e}(g^{x_\sigma} \cdot \prod_{k=\alpha_1}^{\alpha_1, k \neq \gamma} g^{x_k}, \quad g^a)^b) \\
  \end{cases}
\]

\[
= \begin{cases} 
  l + 1, H_5(\hat{e}((g^b)^{x_\sigma}, \quad g^a) \cdot \tau \cdot \hat{e}(\prod_{k=\alpha_1}^{\alpha_1, k \neq \gamma} (g^b)^{x_k}, \quad g^a)) \\
  \end{cases}
\]

It is trivial to compute \( U, V, W \), since the private key \( d_S \) of \( ID_S \) is known \((d_S \) can be obtained by querying RegisterAgent/IssueAttribute queries). After that, sets \( C = (X, U, V, W, ATB\text{-}SET) \) and responds to \( \mathcal{A} \) with \( C \). Here, assume the tuples \((Atb_k, Q_k, x_k)\) (for all \( k, \alpha_1 \leq k \leq \alpha_l \)) already exist on the \( H\text{-}list \). Otherwise, \( \mathcal{B} \) can make queries to obtain these.

**Phase 6:** \( \mathcal{A} \) outputs its guess \( \omega' \leftarrow \{0, 1\} \). If \( \omega' = 0 \), \( \mathcal{B} \) outputs TRUE for the given instance of the DBDH problem; If \( \omega' = 1 \), \( \mathcal{B} \) outputs FALSE for the given instance of the DBDH problem.

**Analysis:** The way for adversary \( \mathcal{A} \) to distinguish \( POL \) from a random simulation policy is to distinguish the part \( ATB\text{-}SET \) in \( C \). Observing \( ATB\text{-}SET \), we can find that: if \( \tau = \hat{e}(g, g)^{abc} \), then \( ATB\text{-}SET \) corresponds to the policy \( POL \); if \( \tau \neq \hat{e}(g, g)^{abc} \), then \( ATB\text{-}SET \) is equal to a random simulation. Therefore, \( \mathcal{B} \) can solve the given instance of the DBDH problem if \( \mathcal{A} \) can distinguish the policy \( POL \) from a random simulation. The probability of \( \mathcal{B} \) solving the given instance of the DBDH problem depends on the following events:

- \( \mathcal{B} \) does not abort during the RegisterAgent/IssueAttribute queries. The probability is at least \((1 - 1/q_h)^{ac} \geq 1/e \). We assume adversary \( \mathcal{A} \) made at most \( q_H \) queries to the \( H \) oracle, at most \( q_C \) RegisterAgent/IssueAttribute queries.
• \( \mathcal{A} \) outputs a correct guess. The probability is at least \( \epsilon \).

Therefore, the probability of \( \mathcal{B} \) solving the given instance of the DBDH problem is

\[
\text{Adv}_B \geq \frac{\epsilon}{e}.
\]

5.4 Summary

In this chapter, we introduced a novel secure multi-agent scheme for information exchange in an untrusted network. Existing works focus only on securing the information exchanged. We argue that only securing the information is not sufficient. Our scheme provides not only information confidentiality but also agent privacy in multi-agent systems. We use attributes to describe the characteristic of the agent. In our multi-agent system, group members can be updated dynamically by applying attributes. Several significant security properties were defined in order to support our scheme. We also defined security model and provided rigorous security proofs in the random oracle model.
In this thesis, we explored security issues and solutions for software agents. We proposed several secure schemes for agent-based transactions and information exchange. We addressed the security issues, such as delegation abuse in a hostile environment, and agent privacy protection during a negotiation in an untrusted environment. Our main contributions in this thesis can be summarised as follows.

In Chapter 3, a secure mobile agent transaction scheme was proposed to allow a mobile agent owner to select remote hosts for authorising a transaction jointly with the mobile agents. The selected hosts list can be updated (i.e. add or delete hosts) dynamically by the agent owner without any additional computation cost. The designated hosts list is embedded into a delegation token whose size is independent of the list. Our scheme eliminates the delegation abuse and the repudiation problems. We also provided the formal security model together with rigorous security proofs to our scheme. Our security model captures the most powerful attacks against a malicious host and a malicious agent owner. The security proofs are based on the hardness of the Computational Diffie-Hellman problem and the q-Strong Diffie-Hellman problem in the random oracle model.

In Chapter 4, we proposed a variant of the mobile agent scheme presented in Chapter 3. Apart from offering the dynamic designated hosts feature, this scheme also reduces the communication overhead without decreasing the security level. A formal definition of the security model associated with the security proofs to the scheme are also provided. The security proofs are based on the Gap Bilinear Diffie-Hellman assumption and the q-Strong Diffie-Hellman assumption in the random oracle model. We also compared the bandwidth and the performance between this scheme and the scheme presented in Chapter 3.

In Chapter 5, we presented a secure information exchange scheme for multi-agent systems in an untrusted network. Compared with the other schemes proposed in
the literature, our scheme provides not only data confidentiality but also privacy. Our scheme is secure against the adaptive chosen ciphertext attack which is the strongest security level. We defined several significant security properties which are not addressed in the literature. Our security proofs are based on the hardness of the Bilinear Diffie-Hellman problem and the Decisional Bilinear Diffie-Hellman problem in the random oracle model.

The three schemes presented in this thesis capture the security requirements of agents. Our second scheme has a very short signature length, which is desirable for agent applications. In the future work, we will conduct the research on how to improve the performance on these schemes. It includes how to reduce the paring operations. We will also work on new schemes which are secure in the standard model rather than in the random oracle model.
Bibliography


[CEdG88] David Chaum, Jan-Hendrik Evertse, and Jerven Van de Graaf. An improved protocol for demonstrating possession of discrete logarithms and


BIBLIOGRAPHY


