Intelligent Information Management, **2010, 2, 262-277**
doi:10.4236/iim.2010.23031 Published Online April 2010 (http://www. SciRP.org/journal/iim)

A Collusion-Resistant Distributed Agent-Based Signature Delegation (CDASD) Protocol for E-Commerce Applications

Omaima Bamasak
Department of Computer Science, College of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia
E-mail: *obamasak@yahoo.co.uk, obamasek@kau.edu.sa*
Received October 21, 2009; *revised December* 22, 2009; *accepted February* 5, 2010

Abstract

Mobile agent technology is promising for e-commerce and distributed computing applications due to its properties of mobility and autonomy. One of the most security-sensitive tasks a mobile agent is expected to perform is signing digital signatures on a remote untrustworthy service host that is beyond the control of the agent host. This service host may treat the mobile agents unfairly, *i.e.* according to its' own benefit rather than to their time of arrival. In this research, we present a novel protocol, called Collusion-Resistant Distributed Agent-based Signature Delegation (CDASD) protocol, to allow an agent host to delegate its signing power to an anonymous mobile agent in such a way that the mobile agent does not reveal any information about its host's identity and, at the same time, can be authenticated by the service host, hence, ensuring fairness of service provision. The protocol introduces a verification server to verify the signature generated by the mobile agent in such a way that even if colluding with the service host, both parties will not get more information than what they already have. The protocol incorporates three methods: Agent Signature Key Generation method, Agent Signature Generation method, Agent Signature Verification method. The most notable feature of the protocol is that, in addition to allowing secure and anonymous signature delegation, it enables tracking of malicious mobile agents when a service host is attacked. The security properties of the proposed protocol are analyzed, and the protocol is compared with the most related work.

Keywords: Agent-Based Signature Delegation, Anonymous Digital Signature, Signature Fairness, Collusion-Resistant Signature

1. Introduction

The widespread of the Internet and the powerful architecture of the World Wide Web (WWW) have transformed the market standards and created many opportunities for conducting business online (*i.e.* e-commerce). Inline with the growth of e-commerce, there have been rapid developments in the area of mobile agent, or software entities that can autonomously perform a given task in open, dynamic and heterogeneous environments. Integrating mobile agents into e-commerce applications (e.g. online shopping and auctioning) to automatically or semi-automatically perform e-commerce tasks makes the Internet reaches its full potential as an electronic marketplace. Users can set up mobile agents and dispatch them to collect product information, process an order, join an auction, pay for an order, deliver the goods, etc, instead of performing the transaction manually.

As agent-based e-commerce technology becomes more developed and standardized, we anticipate that hundreds of mobile agents will be seamlessly embedded in the WWW. Their autonomous nature and heterogeneous interactions among them dramatically reduce the cost and time incurred in performing e-commerce transactions. However, prior to fully enjoy the advantages brought by the mobile agents, the risks and vulnerability they may introduce are also inevitable. Various mobile agents designed by different kind of programmers/ developers can work, interact, and also attack at anytime from anywhere in the web, where the distance is close to null and the transactions can be performed instantly. This has made security an issue that must be considered and

woven into any agent-based e-commerce environment.

One of the most security sensitive tasks a mobile agent encounters in performing an e-commerce transaction is to sign a digital signature on behalf of its owner (*i.e.* agent host) autonomously on a service host. The service host may not be trustworthy; for example, it may attempt to steal the signature key and forge signatures for its own benefit. On the other hand, service hosts, openly providing an execution environment for different kinds of mobile agents, increases the possibility that they may be attacked by malicious agents. In addition, the exposure of the identities of agents and/or agent hosts may lead to unfairness in service provision. For example, in an online auction activity, a service host may favor a particular mobile agent (if its identity is known) and grant it a higher priority in service provision over other mobile agents.

To overcome some of these security problems, a Trusted Third Party (TTP) based approach has been widely used in which a TTP is employed to assist the execution and completion of a electronic transaction (e.g. digital signature protocols) or to resolve any disputes incurred during the transaction process [1-15]. During the execution of a digital signature protocol, the TTP could provide a protection for both the mobile agent and the service host from attacks launched by its counterpart. This is done by facilitating a fair exchange of the signatures between the two signing parties and by preserving the evidence of the transaction. As a mediator, the TTP may have access to the signatures, the signed document, or any related evidence. Therefore, any collusion between the TTP and one of the signing parties will result in undesirable consequences in which the other party will be left in a disadvantage position.

To prevent collusion, and to ensure fairness, in which all the mobile agents are treated equally by the service host [16], mobile agents should be anonymous during the course of a transaction execution. Therefore, how to achieve identity authentication and how to ensure that the agents behaviors are accountable, while, at the same time, preserving identity anonymity of mobile agent, are an open research issue. The scope of the research presented in this paper is to address the above mentioned issues by investigating and designing effective mechanisms that provides a secure and fair mobile agent-based signature delegation environment.

This research addresses this open issue by designing a solution that splits the duties of the TTP, (e.g. partial signature generation and signature verification as presented in [2]), to be undertaken by two separate entities (TTP and Verification Server VS). The separation is designed in such a way that even if the VS and the service host collude, they will not get more information than each party already have. The solution also incorporates blind signature scheme proposed by Chaum [17] to achieve agents' identity anonymity, and hence, facilitate the fairness property mentioned above.

The research presented in this paper is aimed at achieving the following objectives:

1) To identify security requirements for collusion-resistant and fair agent-based signature delegation.

2) To investigate and critically analyze related work in the context of agent security and signature delegation.

3) To advance the state-of-the-art by designing a secure, efficient and viable solution to collusion-resistant and fair signature delegation in the agent environment.

4) To evaluate the security of the designed solution using informal analysis.

5) To prototype the design and evaluate its performance.

This research has made the following advances to the state-of-the-art. It has addressed the mobile agent-based anonymous signature delegation issue by presenting a novel Collusion-Resistant Distributed Agent-Based Signature Delegation (CDASD) protocol, which incorporates three methods as its building blocks, namely, Agent Signature Key Generation method, Agent Signature Generation method, Agent Signature Verification method. The protocol makes use of Chaum's blind signature scheme [17] to allow for a trusted third party (TTP) to blindly certify the mobile agent signature key for mobile agent's anonymity. The mobile agent, while residing at the service host, does not reveal any information about its host's identity, which deprive the service host from favoring a mobile agent on the others, hence, ensuring fairness of service provision. A mechanism is introduced to track down malicious mobile agents and penalize their hosts accordingly. The protocol also provides non-repudiation of signature generation and receipt so that neither the mobile agent and its host nor the service host can deny generating and receiving the signature, respectively. The duties of the TTP, (e.g. partial signature generation and signature verification as presented in [2]), is split to be undertaken by two separate entities (TTP and Verification Server VS). The separation is designed in such a way that even if the VS and the service host collude, they will not get more information than each party already has. Security-sensitive messages exchanged between protocol's parties are signed and encrypted to prevent unauthorized disclosure or tampering with the contents of these messages.

The remaining of this paper is organized as follows: Section 2 provides a literature review and critical analysis of the existing work in the area of fair and secure mobile agent signature delegation. Section 3 captures the requirements for secure and efficient collusion-resistant distributed agent-based signature delegation solution and presents the novel cryptographic building blocks that are used to construct the solution to be presented in the next section. Section 4 presents the design of the collusion-resistant distributed agent-based signature delegation solution, *i.e.* CDASD protocol, by integrating the

cryptographic primitives presented in Section 3. Section 5 provides an analysis of the CDASD protocol against the security requirements listed in Section 3. In order to demonstrate the efficiency of the protocol, a comparison has been conducted with the most related protocol in terms of the computational cost. Section 6 provides an overall conclusions and recommendations for future work.

2. Literature Review

As our work is about delegating signing power to mobile agents and incorporating a TTP to assist in e-transaction execution while preserving the fairness property by using blind signature and anonymous agents, existing work in these areas are reviewed and analyzed in this section.

2.1 Proxy Signatures

A proxy signature is a signature scheme in which an original signer delegates his/her signing capability to a proxy signer. When a receiver verifies a proxy signature, he verifies the signature itself as well as the original signer's delegation. In our context, the original signer is the agent owner, who delegates his signing capability to the mobile agent for it to execute authentic operations in a remote host on behalf of the agent owner.

The concept of proxy signature was first introduced by Mambo *et al.* [27]. They classified proxy signatures based on delegation type as full delegation (giving the original signer's private key itself), partial delegation (issuing a new key pair), and delegation by warrant (issuing a certificate stating the delegation information). Since then, various methods of constructing proxy signatures have been proposed [28-38]. As full delegation is not secure for the agent-based signature delegation and partial delegation is more efficient than delegation by warrant, in this research, we will adopt a proxy signature scheme with partial delegation due to its most relevant to our work.

2.2 Trusted Third Party (TTP)

The benefits of using a TTP to assist electronic transactions (e.g. signature signing) are two-fold [1-15]. Firstly, the TTP can mediate during the execution of a signature exchange protocol so as to achieve a number of security properties such as fairness of transaction outcome and non-repudiation without incurring too much computation costs on the two signing parties. Secondly, it can act as a witness for dispute resolution by preserving evidence of the signing. Broadly speaking, there are two types of TTPs. The first is an online TTP [7,9,14], which is heavily involved in the signature signing process, collecting signatures on a document from respective parties, verifying the signatures, and forwards them to their respective counterparts. The on-line TTP also maintains evidence of the transaction. With this on-line based TTP approach, the TTP has access to all the transaction details and contents, and without the presence of the TTP, signing is not possible. Furthermore, any compromise of the TTP or colluding between the TTP and one of the signing parties may lead to severe consequence to the other party. So the TTP is potentially a performance and security bottleneck.

To reduce the involvement of the TTP in the signature signing process, and to minimize the trust on the TTP, thus overcoming the above mentioned weaknesses, the concept of offline TTP has been proposed [1,3,5,6,10,15]. In the offline TTP based approach, the TTP is only invoked when the signing parties themselves could not reach to a successful completion of the signature signing. In other words, only when a dispute arises, e.g. when one of the parties can not obtain the expected item from the other due to network failures or the other party's misbehavior, the off-line TTP is invoked to recover the necessary information (e.g. signatures) and to assist the exchange to come to a fair completion. The exemplar signature protocols using the off-line TTP-based approach include Bao' protocol [3] that employs the concept of Certificate of Encrypted Message Being a Signature (CE MBS) to convince people that an encrypted message contains a party's signature without revealing the signature. This CE MBS proof can be established by an interactive zero-knowledge proof or a non-interactive proof [11]. This proof is considered as the evidence that can be used to prove the existence of other party's signature. Although the evidence is not a formal type of signature, it discloses significant information. Furthermore, the work presented in [13] describes a method to prevent the offline TTP from gaining the exchanged signatures and the corresponding message to be signed when a dispute occurs between the two parties. The protocol also prevents a party from misusing evidence left during the exchange process. These properties are provided by using two ideas: the secret divide method and the convertible signature.

2.3 Anonymous Agent and Fairness

Another concern in mobile agent based e-commerce environment is maintaining fairness of service provision principle, which is defined in [16] as "the equal treatment of authenticated mobile agents by service hosts". If this fairness principle were followed, service host, such as merchant hosts and auction hosts, would process the requests from authenticated mobile agents according to their time of arrival rather than according to the service hosts' own benefit. Otherwise, the fairness principle is violated and the ideal environment of e-commerce in which authorized participants are assumed to be competing fairly against each other will no

longer be ensured. The work in [16] proposed a mobile agent environment in which the above notion of fairness is preserved as well as offering a protection for service hosts from being attacked by malicious unknown mobile agents. In this work, the blind signature concept proposed by Chaum [17] is utilized to provide agent anonymity for fairness service. A tracking service is also used to penalize the misbehaving mobile agent. However, this approach requires that the mobile agent gets a signed permission from a service host on the services it offers prior to the actual migration for executing its tasks. In addition of being considered as extra communication overhead, this results in the service host's ability to link the mobile agent's identity with its permission and thus, violating the agent anonymity. This approach also extensively use public/private keys for encryption and digital signatures.

The work presented in [18] also provides mobile agent anonymity framework. It uses agent identity encryption to hide the identity of the agent and controls access to services and resources by allocating privileges based on partially blind signature. However, this scheme does not provide a mean to track down and penalize a misbehaving anonymous mobile agent.

3. The Design of Secure and Anonymous Mobile Agent-Based Signature Delegation Building Blocks

This section presents our novel cryptographic building blocks that are used to construct our secure and efficient collusion-resistant distributed agent-based signature delegation solution, *i.e.* the CDASD protocol, to be presented in the next section. These cryptographic building blocks are: Agent Signature Key Generation method, Agent Signature Generation method, and Agent Signature Verification method. In detail, Subsection 3.1 specifies the security requirements for fair signature generation process performed by a mobile agent in a service host. Subsection 3.2 outlines design principles for the cryptographic primitives and the assumptions on which the design is based. Subsection 3.3 gives the notations used in the description of the cryptographic building block and the protocol and a brief description of Chaum's blind signature scheme. Subsection 3.4 presents a detailed description of the proposed cryptographic building blocks.

3.1. Requirements Specification

As this paper describes the design of a Collusion-Resistant Distributed Agent-Based Signature Delegation (CDASD) protocol, the following lists security and functional requirements the CDASD protocol is aimed at satisfying.

3.1.1. Security Requirements

S1) Verifiability of the signature: Validity of the signature generated by the mobile agent on a document *M* can be verified using public parameters.

S2) Unforgeability of the signature: It is difficult for any other entities than the agent's owner and the agent itself to generate a valid signature on the specified document.

S3) Non-repudiation of signature origin: It is difficult for an original signer (*i.e.* the agent host) to falsely deny that it has delegated the signing power to the agent.

S4) Non-repudiation of signature receipt: It is difficult for a signature recipient (*i.e.* the service host) to falsely deny that it has received the signature, if this signature is taken as the proof of a deal conducted by the mobile agent and the recipient.

S5) Collusion-resistance: it should be difficult for the *VS* and the service host, if collude together, to get any advantage over a mobile agent and the agent host.

S6) Unlinkability: Deciding whether two different valid signatures were computed by the same mobile agent is computationally hard.

S7) Anonymity: The real identity of a mobile agent should not be revealed to any party other than the agent host itself.

S8) Fairness of service provision: The service host should only process requests made from authenticated mobile agents and on the first-come-first-serve basis.

S9) Agent Host and Mobile agent Accountability: Any misbehavior by a mobile agent should be detectable and its host will be accounted for.

3.1.2. Other Requirements

P1) Protocol efficiency: The computational and communication overheads introduced as the result of using the distributed approach to the role of the *TTP* should be kept as low as possible.

3.2. Design Principles

The design of our CDASD protocol is based upon the following hypothesis, *i.e.* if the service provider, *i.e.* TTP/verification service and service host, can not link a request for the service to the identity of the service requestor (*i.e.* a mobile agent or agent host), then it would be more difficult, or less likely, for the service provider to collude with any of the service requestors to gain unfair advantages over other service requestors.

To realize the above mentioned hypothesis, a number of design principles, *i.e.* measures, have been taken into account in the protocol design. They are listed in the following:

- **Measure 1**. The mobile agent signature key is generated in such a way that it does not reveal any information about the agent host or the mobile agent identities. It is also one-time, *i.e.* a key is used to generated only one

signature. This supports the unlinkability of signatures property.

- **Measure 2**. The blind signature scheme proposed by Chaum [17] is used in our protocol to allow for the *TTP* to blindly certify the mobile agent signature key without having knowledge neither of the key nor of the mobile agent's identity. Thus, supporting anonymity of the mobile agent. However, the service host needs to authenticate the arriving mobile agents so as to provide them with the services they request. To solve this dilemma, *i.e.* making the mobile agent anonymous and, at the same time, can be authenticated, the *TTP* (the party that is trusted by all other parties of the protocol) certifies, *i.e.* signs, the agent's signature key. Thus, the service host will use this signature as a mean to authenticate the mobile agent.
- **Measure 3.** The signature generated by the mobile agent can only be verified by the verification server through the use of a commitment generated by the agent host, rather than the agent host's public key corresponding to the signature key used to generate a conventional signature. By doing so, we deliberately deprive the service host from this signature verification capability in order to achieve non-repudiation of service requests and provisions.
- **Measure 4.** A penalty system is applied on a misbehaving mobile agent and its host. After each transaction is completed, the service host assigns a feedback flag to the transaction and sends it to the *TTP*. The values given are dependent on the outcome of the transaction, *i.e.* signature generation process, performed by the mobile agent. For example, if the transaction outcome is positive, the flag value will be 'Success'; if the outcome is negative due to the signature not passing the verification process, then the flag value will be 'Attack'; and if the outcome is negative due to any other reasons, then the flag value will be 'Failure'. The *TTP*, upon receiving the outcome value, updates the status of the corresponding *AH*. That is, if the *TTP* receives 'Attack' as an outcome flag, it will increment agent host's associated counter of malicious incidence. When this counter reaches a certain threshold specified by the *TTP*, *i.e.* five incidences, this agent host will be blacklisted and the *TTP* will refuse to provide it with any service in the future. This measure will deter the agent host from sending mobile agents to service hosts for malicious purposes.

3.3. Preliminaries

In this section, we outline the notion used in the protocol description and the assumptions on which the protocol is designed. This is followed by outlining Chaum's blind signature scheme as it is incorporated in the generation and certification of a mobile agent's signature key to facilitate agent anonymity.

3.3.1. Notation

- $H(x)$ is a one-way collision-free hash function that takes a variable sized input (x) and produces a fixed-size output (digest). It should have the following properties: (1) for any x, it is easy to compute $H(x)$; (2) given x, it is hard to find x' ($\neq x$) such that $H(x) = H(x')$; and (3) given $H(x)$, it is hard to compute x. SHA-1 [20] is an example of such a one-way hash function.
- Sign($\{d_I, n\}$, M) denotes a signature of party I on an item M (e.g. a hash value of a document) using the RSA signature scheme [19] with a private key $\{d_I, n\}$ of I. RSA is based on two large prime numbers (p and q), which are multiplied together to get the public modulus n. Party I calculates $f(n) = (p-1)(q-1)$ and chooses e_I to be relatively prime to $f(n)$ and less than $f(n)$. Party I then determines d_I such that $d_I \times e_I = 1 \ mod \ f(n)$ and $d_I < f(n)$. The public key is $\{e_I, n\}$ and the private key is $\{d_I, n\}$. The signature of party I on message M with its private key is expressed as $\text{Sign}(\{d_I, n\}, M) = H(M)^{d_I} \bmod n$.
- Verify($\{e_I, n\}$, S_I, M) denotes the result of the verification of party I's RSA signature $S_I = \text{Sign}(\{d_I, n\}, M)$ on M with I's public key $\{e_I, \text{n}\}$. To verify the signature, the receiver first computes the hash value of *M' received*, $H(M')$, and then calculates $T = S_I^{e_I} \bmod n = H(M)^{e_I \times d_I} \bmod n = H(M)$ It then compares $H(M')$ with T and, if the two values are equal, the signature is considered valid, which is expresses as Verify($\{e_I, n\}$, S_I, M) = true, otherwise, Verify($\{e_I, n\}$, S_I, M) = false.
- $E(\{e_I, n\}, M)$ denotes an encryption of item M using the RSA encryption scheme [19] with the public key $\{e_I, n\}$ of party I.
- $D(\{d_I, n\}, M)$ denotes a decryption of item M using the RSA encryption scheme [19] with the private key $\{d_I, n\}$ of party I.
- $A \xrightarrow{E} B$: m denotes that party A sends a message m to party B via an external channel such as a telecommunication network.
- $A \xrightarrow{I} B$: m denotes that party A sends a message m to party B via an internal message passing mechanism. This case applies when both A and B resides at the same host.
- ID_I, $I \in \{AH, SH, MA, TTP, VS\}$, denotes party I's unique identity, where *AH* denotes Agent Host, *SH* service host, *MA* mobile agent, *VS* verification server.

3.3.2. Assumptions

- Every party or host $I \in \{AH, SH, TTP, VS\}$ participating in the protocol execution has a pair of RSA public and private keys $\{e_I, n\}$ and $\{d_I, n\}$, as defined in Subsection 3.3.1. The public key $\{e_I, n\}$ is certified in the form of a digital certificate *Cert*(I) signed by a certification authority (*CA*), which is trusted by all parties.
- Parties *AH* and *SH* have each *TTP*'s and *VS*'s public key certificate *Cert(I)*. The *TTP* and *VS* also have the

public key certificates of each other and of the parties participating in the protocol, *i.e. AH* and *SH*. These certificates will play a role in authentication and secure communications between these parties.

- Parties *AH* and *SH* may not have mutual trust. That is, either of them may misbehave in order to gain some advantages over the other party. For example, party *SH* may try to use *MA*'s signature key to sign more that one deal for which *AH* (*i.e.* the user) will be held responsible. *TTP* and *VS* are introduced in the protocol to assist *MA*'s signature verification and to store transaction evidence for dispute resolution. It is assumed that *TTP* and *VS* will not misbehave or collude with each other or with any other party.
- *Req* represents the service required by *AH* which *MA* is delegated to perform on service host *SH*. For example, *Req* typically includes service name, validity period and transaction-specific information (e.g. good type, price, etc).
- Party *AH* (Agent Host) delegates his mobile agent *MA* to perform some tasks and sign a document *M* on a service (remote) host *SH*. Typically, *M* is the service (e.g. offer) presented by *SH* that conforms to *Req*.

SH is assumed to provide mechanism to protect the mobile agents it hosts from being eavesdropped on their contents and execution flows by other agents hosted also by *SH*. *SH* can use existing solutions, e.g. tamper-resistant hardware [22] and time limited blackbox security [23], to provide such mechanisms.

3.3.3. Blind Signatures for Untraceable Payments

Chaum proposed a blind signature scheme [17] for untraceable payments based on the RSA public-key cryptographic system [19]. The signature scheme allows a person to get a message signed by another party without revealing any information about the message to the signing party. The scheme works as follows. Assume that Alice has a message *M* on which she wishes to have Bob's signature, and Alice does not want Bob to learn anything about *M* during the signing process. Let {(*e*, *n*) (*d, n*)} be Bob's public and private keys, respectively. The scheme defines the following steps to generate a blind signature on *M*:

1) Alice generates a random number *r* such that gcd (*r*, *n*) = 1, produces a message digest *H*(*M*) for message *M* using a hash function *H*(), and sends $x = r^e \times H(M) \ (mod\ n)$ to Bob. The value of *H(M)* is "blinded" by the random value *r*, hence Bob can derive no useful information from it.

2) Bob signs *x* using his private key and return the signed value $t = x^d \ (mod\ n)$ to Alice.

3) Since $x^d = (r^e \times H(M))^d = r \times H(M)^d \ (mod\ n)$, Alice can obtain Bob's signature *S* on *M* by "unblinding" the value *t* by computing $S = t\, r^{-1} \ (mod\ n)$.

3.4. The CDASD Protocol Building Blocks

The Collusion-Resistant Distributed Mobile Agent-Based Signature Delegation (CDASD) protocol is built upon three novel cryptographic methods: the Agent Signature Key Generation Method, the Agent Signature Generation Method, and the Agent Signature Verification Method. The protocol is initiated by the user (represented by an *AH*) who executes, in cooperation with the *TTP*, the Agent Signature Key Generation Method, to generate a certified mobile agent signature key S_{MA} using agent's anonymous ID ($Anony\text{-}ID_{MA}$) and Chaum's blind signature scheme. Using the Agent Signature Generation Method and the signature key, S_{MA}, the mobile agent generates a signature on an offer made by a service host, *SH*. Once the signature is generated, (only) the Verification Server, *SV*, can verify the correctness of the signature by using the Agent Signature Verification Method. These three methods are described in detail below.

3.4.1. Agent Signature Key Generation Method

The Agent Signature Key Generation Method is executed by the *AH* with the assistance of the *TTP* to generate its signature key S_{MA}. In addition, as mentioned in Subsection 3.2 (Design principles), we have devised an idea of using a commitment generated by the signature key generator *AH* (instead of using its public key) for the agent signature verification, The following gives the details as how the signature key S_{MA} and the commitment $Comm_{MA}$ are generated by *AH*. The commitment $Comm_{MA}$ will be used by the signature verifier (*VS*) to verify the signature to assure that $\text{Sign}_{\text{MA}}(Doc)$ has indeed been generated by using the correct signature key S_{MA}, the signature is generated only once using the signature key S_{MA}, and that the signed document meets the user's requirements *Req*. For an *AH* to generate an agent signature key and the corresponding commitment, it performs the following calculations.

1) The agent host, *AH*, first generates an anonymous identity ($Anony\text{-}ID_{MA}$) for the mobile agent *MA*.

2) *AH* then generates a random number *r*, uses *r* to blind the hash value of the agent's anonymous identity and sends it to the *TTP* after being encrypted with *TTP*'s public key (as Chaum's blind signature algorithm). That is,

$$Z = H(Anony - ID_{MA}) \times r^{e_{TTP}}$$

3) *TTP*, upon the recipient of the request from *AH*, blindly signs *Z*, and sends it back to *AH*.

$$T = Z^{d_{TTP}} = H(Anony - ID_{MA})^{d_{TTP}} \times r^{e_{TTP} \times d_{TTP}}$$
$$= H(Anony - ID_{MA})^{d_{TTP}} \times r$$

Here, *TTP* has signed *Z* without knowing its contents.

4) *AH* unblinds *T* to reveal *MA*'s signature key S_{MA}:

$$S_{MA} = T / r = (H(Anony - ID_{MA})^{d_{TTP}} \times r) / r$$
$$= H(Anony - ID_{MA})^{d_{TTP}}$$

S_{MA} is the signature key to be used by *MA* to sign documents on *SH* on behalf of *AH*. It can be seen that S_{MA} represents *TTP*'s signature on *MA*'s anonymous identity.

5) *AH* also constructs a commitment $Comm_{MA}$ containing four items: hashed *MA*'s anonymous ID $H(Anony\text{-}ID_{MA})$, *Bond* (to be described in Subsection 4.1), *Req*, and the key's validity period (lifetime), signed with *AH*'s private key (*i.e.* Sign($\{d_{AH}, n\}$, $H(Anony\text{-}ID_{MA})$, *Req*, *Lifetime*) = $H(H(Anony - ID_{MA}), \mathrm{Re}q, lifeti\text{-}me)^{d_{AH}} \bmod n$). When *AH* dispatches *MA*, it sends $Comm_{MA}$ to *VS* for signature verification purpose.

3.4.2. Agent Signature Generation Method

MA, residing at *SH*, generates a signature on document *Doc* using S_{MA}, *i.e.* $D = S_{MA}^{H(Doc)} \bmod n$, where $\mathrm{Sign}_{\mathrm{MA}}(Doc) = (Doc, D)$ is *MA*'s signature on *Doc*.

3.4.3. Agent Signature Verification Method

The signature is verified by a Verification Server (*VS*) using the method described below.

1) When *SH* wants to verify the signature $\mathrm{Sign}_{\mathrm{MA}}(Doc)$ generated by *MA*, it sends *Doc* signed with its private key (*i.e.* Sign($\{d_{SH}, n\}$, *Doc*) = $H(Doc)^{d_{SH}} \bmod n$) together with *MA*'s signature on *Doc* (*i.e.* $\mathrm{Sign}_{\mathrm{MA}}(Doc) = (Doc, D)$) to *VS*.

2) Upon the receipt of the these items, *i.e.* (Sign($\{d_{SH}, n\}$, *Doc*) || $\mathrm{Sign}_{\mathrm{MA}}(Doc)$), where || denotes concatenation, *VS* performs the following computations:

a)
$$T = D^{e_{TTP}} \bmod n = S_{MA}^{e_{TTP} \times d_{TTP} \times H(Doc)} \bmod n$$
$$= H(Anony - ID_{MA})^{H(Doc)} \bmod n$$

b) Computes the hash of *Doc* received in $\mathrm{Sign}_{\mathrm{MA}}(Doc)$, (*i.e.* $H(Doc)$), uses this freshly computed hash value together with the hash value of the *MA*'s anonymous ID received earlier from *AH* (*i.e.* $H(Anony\text{-}ID_{MA})$) to compute $Y = H(Anony\text{-}ID_{MA})^{H(Doc)} \bmod n$.

c) Check if $T = Y$; if positive, then the signature $\mathrm{Sign}_{\mathrm{MA}}(Doc)$ is valid.

4. The Collusion-Resistant Distributed Agent-Based Signature Delegation (CDASD) Protocol

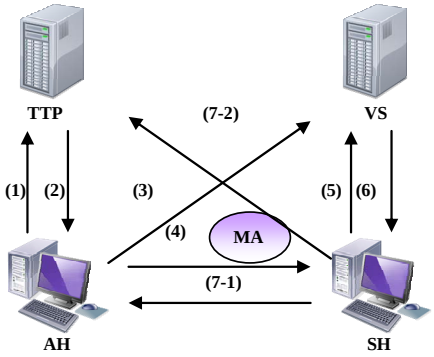
This section presents the design of the CDASD protocol by integrating the cryptographic primitives presented in Section 3. In detail, Subsection 4.1 gives an overview of the protocol's environment. Subsection 4.2 presents a detailed description of the CDASD protocol's steps.

4.1. Protocol Overview

The protocol consists of five types of players, Agent Hosts (*AH*), Mobile Agents (*MA*), Service Hosts (*SH*), a Trusted Third Party (*TTP*), and a Verification Server (*VS*), as shown in **Figure 1**. The roles played by each of the players are detailed below.

1) An Agent Host (*AH*) performs the following three tasks. Firstly, it captures and records the user's shopping requirements, generates an anonymous ID for the mobile agent ($Anony\text{-}ID_{MA}$), and blinds it before sends it to the *TTP* for signature signing (*i.e.* for certification). Secondly, *AH* generates a *Bond* for the mobile agent, which is the hash value of the concatenation of a random number and $Anony\text{-}ID_{MA}$, that is, $H(\mathrm{rand}||Anony\text{-}ID_{MA})$. This *Bond* maps to one and only one mobile agent ID (ID_{MA}). Once being sent to the *TTP*, the *Bond* will act as the *MA*'s pseudonym and will be used to track down and penalize a misbehaving *MA*. The third task is to initiatethe mobile agent and dispatch it to the *SH* that will prservice the *AH* requests. To accomplish the above tasks, the *AH* has to record the user's shopping requirements, ovide the the certified signature key, the blinding factor, the Bond and the lifetime, which is the validity period of the certified signature key. **Table 1** shows 'Transaction Information' containing the above mentioned items.

2) A Mobile Agent (*MA*) is delegated by an *AH* to accomplish certain tasks on his behalf. This includes searching *SH*s for the service required by *AH*, signing a suitable offer with the certified signature key (S_{MA}), and



(1) MA signature key request.
(2) MA signature key delivery
(3) Verification-aiding items
(4) MA dispatch
(5) MA signature verification request
(6) MA signature verification response.
(7-1) MA return
(7-2) Transaction outcome

Figure 1. The outline of the CDASD Protocol.

returning back to *AH*.

3) A Service Host (*SH*) provides various services, for example, data retrieving, providing product information, selling goods, etc.

4) The Trusted Third Party (*TTP*) is responsible for 'blindly' certifying a *MA*'s signature key. *TTP* does not have any knowledge of the signature key, thus preserving the anonymity of the mobile agent and preventing collusion between the *TTP* and the *SH* (further details can be found in Subsection 5.2). In addition, *TTP*'s signature on the key gives *SH* confidence that the key has come from a trusted source although the entity carrying and representing the key (*MA*) is anonymous. *TTP* is also responsible for identifying a misbehaving *MA* and then penalizing the corresponding *AH*. For doing so, *TTP* maintains two tables: the AH-MA Relation table and the AH Trust table. The AH-MA Relation table records *AH*'s identity, *MA*'s certified signature key, the key validity period, the Bond, and the Status. The Status field records the execution result of the protocol, e.g. 'Success', 'Failure', or 'Malicious Attack' on the visiting *SH*s. The *AH* Trust table records the count of each *AH*'s malicious behavior. In other words, it records how many mobile agents coming from the same *AH* behave maliciously. As the *TTP* records each *MA*'s behavior in the Status field of the AH-MA Relation table, the *TTP* can count how many *MA*s with the same *AH* acts maliciously and then record the count result in the respective field in the AH Trust table. The *TTP* decides an *AH* to be blacklisted, and hence refuse to provide any service to it, when the count reaches to a certain threshold (E.g. 5 attempts). **Figure 2** shows AH-MA Relation table and AH Trust table.

5) The Verification Server (*VS*) is responsible for verifying a signature signed by a *MA* using its certified signature key (S_{MA}) with an aid of a commitment $Comm_{MA}$ that is sent earlier by *AH*. The *VS* maintains a table named "Verification Information" containing the data necessary for signature verification process together with the verification outcome: the hashed *MA*'s anonymous ID ($H(Anony\text{-}ID_{MA})$), the *Bond* value, the user shopping requirements *Req*, the validity period of the usage of the certified signature key S_{MA} (*Lifetime*), the signature to be verified $Sign_{MA}(Doc)$, *SH*'s signed request $Sign(\{d_{SH}, n\}, Doc)$, and the verification result (Pass/Fail). The table is depicted in **Table 2** below.

4.2. Protocol Description

This section describes the protocol designed using the methods presented in Section 3. The protocol consists of three phases: Certified Signature Key Acquisition, Service Request & Signature Generation, and Signature Verification.

Phase 1: Certified Signature Key Acquisition

In this phase, the *AH* initiates a protocol run by capturing the user's shopping requirements, generates a signature key for the mobile agent *MA* and gets it certified anonymously by the *TTP*. The detailed description of Phase 1 is depicted as follows:

Step 1: The *AH* captures the user' shopping requirements *Req*. *AH* then executes steps 1 and 2 of the Signature Key Generation method (Subsection 3.4.1) to generate an anonymous agent ID ($Anony\text{-}ID_{MA}$) and blind it with the randomly generated number *r*. It then encrypts the result, $Z = H(Anony - ID_{MA}) \times r^{e_{TTP}}$, with *TTP*'s public key. *AH* also generates a *Bond* for a mobile agent, which is the hash value of the concatenation of a random number with $Anony\text{-}ID_{MA}$, that is $H(rand||Anony\text{-}ID_{MA})$. The *AH* then initiates a certified signature key request (Cert-Key) message that contains *AH*'s identity, *Z*, *Bond*, the signature key validity period *Liftime*, and *Req*. The message is then signed with *AH*'s private key, encrypted with *TTP*'s public key, and sent to the *TTP*:

T1: $AH \xrightarrow{E} TTP$: E ($\{e_{TTP}, n\}$, (Sign($\{d_{AH}, n\}$, Cert-Key))

Where, Cert-Key = $\{ID_{AH}, Z, Bond, Lifetime, Req\}$.

Table 1. Transaction Information table maintained by *AH*.

ID_{MA}	Anony-ID_{MA}	Blinding factor (r)	Requirement (Req)	Certified signature key (S_{MA})	Bond	Lifetime

Table 2. Verification Information Table Maintained by *VS*.

$H(Anony\text{-}ID_{MA})$	Bond	Req	Lifetime	$Sign_{MA}(Doc)$	$Sign(\{d_{SH}, n\}, Doc)$	Verification Result

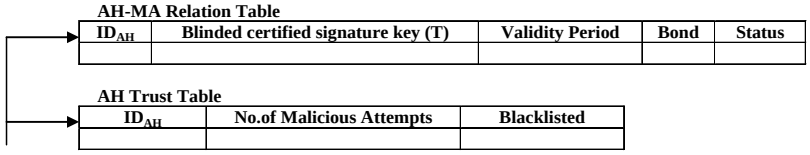


Figure 2. AH-MA Relation Table and AH Trust Table Maintained by TTP.

Step 2: For the signature verification purpose, *AH* executes step 5 of Agent Signature Key Generation method (Subsection 3.4.1), *i.e.* generates a commitment $Comm_{MA}$ = Sign({d_{AH}, n}, H(*Anony-ID*$_{MA}$), *Bond, Req, Lifetime*), encrypts it with *VS*'s public key for confidentiality, and sends it to *VS*.

T2: $AH \xrightarrow{E} VS$: E ({e_{VS}, n}, $Comm_{MA}$)

Step 3: Once the encrypted and signed Cert-Key message from *AH* in **T1** is received, the *TTP* first decrypt the message using its private key, *i.e.* D({d_{TTP}, n}, (Sign({d_{AH}, n}, Cert-Key)), and then performs the following verification:

Verification TTP-1:

a. Check the correctness of *AH*'s signature on Cert-Key using *AH*'s public key as described in Subsection 3.3.1:Verify({e_{AH}, n}, Sign({d_{AH}, n}, Cert-Key)

b. Check, if ID_{AH} exists in the AH Trust table, that it is not 'blacklisted'.

If the outcome of any steps of Verification TTP-1 is negative, *TTP* terminates the protocol run and sends an error message to *AH* stating the reason for protocol termination. If Verification TTP-1 (a) is positive and ID_{AH} does not exist in both AH-MA Relation and AH Trust tables (*i.e.* first request), the *TTP* creates a new record for ID_{AH} in both tables and stores the contents of Cert-Key message in the AH-MA Relation table. If Verification TTP-1 (a) is positive and ID_{AH} exists in both AH-MA Relation and AH Trust tables, *TTP* updates table AH-MA Relation with the contents of the new Cert-Key message. In both the latter cases, *TTP* certifies the signature key *Z* by blindly signing *Z* with its private key, encrypts it with *AH*'s public key, and performs transaction **T3**. That is,

Step 4: *AH*, upon the receipt of message **T3**, decrypts its contents using its private key, *i.e.* D({d_{AH}, n}, E({e_{AH}, n}, (Sign{d_{TTP}, n}, *Req*, *Bond*, *Lifetime*), *T*), and then performs the following tasks:

a) Check the correctness of *TTP*'s signature on *Req* using *TTP*'s public key as described in Subsection 3.3.1: Verify({e_{TTP}, n}, Sign({d_{TTP}, n}, *Req*).

b) Unblinds *T* to reveal *MA*'s signature key S_{MA}, *i.e.*

$$S_{MA} = T / r = (H(Anony - ID_{MA})^{d_{TTP}} \times r) / r$$
$$= H(Anony - ID_{MA})^{d_{TTP}}$$

S_{MA} is the signature key to be used by *MA* to sign documents on *SH* on behalf of *AH*. S_{MA} is also *TTP*'s signature on *MA*'s anonymous identity. S_{MA} reveals neither *AH*'s nor *MA*'s identities, thus, preserving the anonymity of *MA* and its source (*AH*) and hence supports the fairness property.

Phase 2: Service Request & Signature Generation

Prior to a service requisition, a user would need to know what services are provided by which *SH*s. It is assumed in this protocol that there is a public directory (e.g. yellow pages) server responsible for the discovery of services provided by *SH*s. Therefore, users (or their respective *AH*s) can discover which *SH*s performs what services via the directory service.

So at this phase, the *AH* sets up a mobile agent *MA* and dispatches it to the corresponding *SH*(s) where the *MA* will generate a signature on the document representing the required service. This process can be described in three steps as follows:

Step 5: The user creates a mobile agent *MA* on his *AH* and supplies it with the following four items: (1) the itinerary (*i.e.* identities or IP address(s) of the *SH*(s) to be visited); (2) *Req* signed by the *TTP*, received in **T3**; (3) the certified signature key S_{MA}, and (4) the *Bond*. *AH* then dispatches *MA* in to the network to migrate to the the *SH*s, and to ask the *SH*s for services that meet *Req*. The *AH* records mobile agent's true identity ID_{MA}, anonymous identity *Anony-ID*$_{MA}$, blinding factor *r*, user requirements *Req*, certified signature key S_{MA}, *Bond*, and *Lifetime* in the related fields in the Transaction Information table.

T4: $AH \xrightarrow{E} SH : MA$

Step 6: Upon the receipt of **T4**, *SH* provides *MA* with an execution environment to execute its task. *MA* starts its execution by sending the following message to *SH*:

T5: $MA \xrightarrow{I} SH$:
$Sign(\{d_{TTP}, n\}, \mathrm{Re}q, S_{MA}, Bond, Lifetime)$

Upon receipt of **T5**, *SH* performs the following verification:

Verification SH-1:

a) Check the correctness of *TTP*'s signature on **T5** using *TTP*'s public key as described in Subsection 3.3.1: Verify({e_{TTP}, n}, Sign({d_{TTP}, n}, *Req, S_{MA}, Bond, Lifetime*).

b) Check the user requirements specified in *Req* to assess if *SH* is able to provide the service (*i.e.* an offer) that conforms to *Req*.

c) Check if the arrival time of **T5** is within the validity period *Lifetime*.

The purpose of Verification SH-1 is to authenticate the mobile agent *MA* and to ensure accountability. For example, if *AH* claims that the *Offer* (see **T6** next) does not conform to the requirements specified in *Req* after the transaction is completed, *SH* can produce the signed *Req* to resolve the dispute. If the outcome of any part of Verification SH-1 is negative, *SH* would terminate the protocol run and send an error message to *MA*. Otherwise, if all parts of Verification SH-1 are positive, *SH* signs the *Offer* using its private key and give it to the authenticated *MA*, *i.e.*

T6: $SH \xrightarrow{I} MA : \begin{cases} Sign(\{d_{SH}, n\}, Offer) \\ Error\ Message \end{cases}$

Step 7: *MA* in this stage will either have received an error message or the signed *Offer*. If the error message is received, *MA* will migrate to the next *SH* in its itinerary to continue its search for a suitable offer. Otherwise, *MA* performs the following verification:

Verification MA-1:

a) Check the correctness of *SH*'s signature on *Offer* using *SH*'s public key as described in Subsection 3.3.1: Verify($\{e_{SH}, n\}$, Sign($\{d_{SH}, n\}$, *Offer*).

b) Confirm that *Offer*'s details comply with the requirements specified in *Req*.

A negative outcome of Verification MA-1 means that *SH*'s service (*Offer*) is not acceptable. As a result, *MA* will stop the protocol execution and move to the next *SH* in its itinerary. If the outcome of Verification MA-1 is positive, *MA* will execute the Agent Signature Generation Method (described in Subsection 3.4.2) to sign the Offer, *i.e.*, *MA* generates a signature on document *Doc* = (*Req*, *Offer*) using S_{MA} by first computing $D = S_{MA}^{H(Doc)} \bmod n$. The *MA*'s signature $\text{Sign}_{\text{MA}}(Doc) = (Doc, D)$. *MA* sends its signature to *SH*.

T7: $MA \xrightarrow{I} SH : Sign_{MA}(Doc)$

It can be noticed from the above steps that *SH* cannot link any *MA* to its *AH* using the information carried by *MA*. The *MA* is authenticated by *TTP*'s signature on its information. Therefore, only authorized (*i.e.* authenticated) *MA* could be served by *SH*, and this service provisioning is undertaken without exposing *MA*'s identity thus achieving fairness in service provisioning.

Phase 3: Signature Verification

In this phase, *SH* verifies the signature of *MA* on the document *Doc* with the help of the Verification Host *VS*. This phase consists of the following steps:

Step 8: Upon receipt of *Doc* signed with S_{MA} in **T7**, *SH* forwards this signature to *VS* together with *Doc* signed with its private key. This is because *SH* cannot verify *MA*'s signature as it does not have the corresponding information needed for the verification. We deliberately deprive *SH* from this signature verification capability in order to achieve non-repudiation of service requests and provisions.

T8: $SH \xrightarrow{E} VS : Sign(\{d_{SH}, n\}, Doc), Sign_{MA}(Doc)$

Step 9: When *VS* receives **T8**, it will perform Verification VS-1:

Verification VS-1:

a) Check the correctness of *SH*'s signature on *Doc* using *SH*'s public key as described in Subsection 3.3.1: Verify($\{e_{SH}, n\}$, Sign($\{d_{SH}, n\}$, *Doc*)).

b) Check the correctness of *MA*'s signature on *Doc* using *TTP*'s public key and the $H(Anony\text{-}ID_{MA})$ received in commitment $Comm_{MA}$, as described in Agent Signature Verification Method (Subsection 3.4.3, step 2).

c) Confirm that *Doc* signed with *SH*'s private key is identical to *Doc* signed with *MA*'s certified signature key S_{MA}.

d) Fetch *Bond* value in the Verification Information table (received earlier (in $Comm_{MA}$) in **T2**) that match the one received in **T8**, then check that the Verification Result field is empty and that the corresponding *Req* in the table conforms to that in both *Docs* mentioned above.

e) Check if the arrival time of **T8** is within the validity period *Lifetime* received in $Comm_{MA}$.

The purpose of Verification VS-1 is to twofold. Firstly, to authenticate *MA*'s signature $\text{Sign}_{\text{MA}}(Doc)$. That is, to ensure that the signature has been generated using the appropriate key (certified by the *TTP*). Secondly, to ensure non-repudiation. That is, *SH* cannot later falsely deny having received *Doc* from *MA* thus refusing to provide the required service, and *MA* cannot later falsely deny that it has being served by *SH*. Finally, to ensure the request freshness, *i.e.* the key is used only once to generate one signature.

If the outcome of any step of Verification VS-1 is negative, *VS* terminates the protocol run, put a "Fail" flag in the Verification Result field of the Verification Information table, and sends an error message to *SH* stating the reason for the protocol termination. If Verification VS-1 is all positive, *VS* stores the items Sign($\{d_{SH}, n\}$, *Doc*), $\text{Sign}_{\text{MA}}(Doc)$, and a "Pass" flag in the Verification Result field. To acknowledge the successful signature verification, *VS* sends $\text{Sign}_{\text{MA}}(Doc)$ back to *SH* signed with its private key, *i.e.*

T9: $VS \xrightarrow{E} SH: \begin{cases} Sign(\{d_{VS}, n\}, Sign_{MA}(Doc)) \\ Error\ Message \end{cases}$

Step 10: *SH*, upon the receipt of the signed *Doc* from *VS*, performs the following verification:

Verification SH-2:

Verify *VS*'s signature on $\text{Sign}_{\text{MA}}(Doc)$ using *VS*'s public key as described in Subsection 3.3.1: Verify($\{e_{VS}, n\}$, Sign($\{d_{VS}, n\}$, $\text{Sign}_{\text{MA}}(Doc)$).

If this verification outcome is positive, *SH* will store a copy of **T9** (*i.e. MA*'s signature approved by *VS*'s signature) and sends a copy to *MA* declaring that this transaction is successful. Otherwise, *SH* sends an error message stating that the transaction has failed.

T10-1: $SH \xrightarrow{I} MA: \begin{cases} Sign(\{d_{VS}, n\}, Sign_{MA}(Doc)) \\ Error\ Message \end{cases}$

In the case where a transaction with a *SH* has failed, *MA* may either continue its journey by migrating to next *SH* in its itinerary or return back to its *AH*.

As a response to **T9**, *SH* sends to the *TTP* a signed and encrypted message containing the corresponding *MA*'s *Bond* and one of the following transaction outcome flags:

a) 'Success'-if **T9** contains the signed *Doc* (Sign($\{d_{VS}, n\}$, $\text{Sign}_{\text{MA}}(Doc)$).

b) 'Attack'-if **T9** contains an error message as a result of not passing Verification VS-1 step (b), this means that *MA* did not generate a correct signature, thus indicating a malicious intention. This acknowledgment is also sent if any attack is launched by *MA* against *SH*.

c) 'Failure'-if the error message is for any other reason.

T10-2: $SH \xrightarrow{E} TTP : E(\{e_{TTP}, n\}, Sign(\{d_{SH}, n\}, Bond, (Success\ or\ Attack\ or\ Failure))$

Step 11: Upon the receipt of message **T10-2**, the *TTP* decrypts the message, *i.e.* $D(\{d_{TTP}, n\}, E(\{e_{TTP}, n\}, \text{Sign}(\{d_{SH}, n\}, Bond, (Success, Attack, \text{or } Failure))$, then updates AH-MA Relation table with the received information. That is, it will fetch *Bond* value and records "Success", "Attack", or "Failure" in the corresponding Status field. If 'Attack' is recorded in the Status field, *TTP* will increment the value in the corresponding 'Number of Malicious Attempts' field of *AH* Trust table. If this value reaches a certain threshold specified by the *TTP* (e.g. five attempts), then *AH* may classify this *MA* as 'Blacklisted'.

The CDASD protocol is formally presented in **Figure 3**. A summary of the methods/algorithms performed by each party in the protocol with their inputs and outputs is given in **Table 3**.

5. The Evaluation of CDASD Protocol

This section presents an analysis of the protocol against the security requirements listed in Subsection 3.1. In order to demonstrate the efficiency of the protocol, a comparison has been conducted with the most related protocol in terms of the computational cost.

5.1. Comparison with Related Work

To highlight the merits of our CDASD protocol, the efficiency, reliability and accountability of our protocol is compared with that of the most related protocol proposed in [16], hereafter referred to as Lin's protocol. The reason for choosing this protocol is that it is designed to perform similar task to ours. That is, an agent host delegates a mobile agent to perform a task on or get a service from a (remote) service host on behalf of the agent host. The mobile agent is made anonymous, *i.e.* does not reveal any information about its or its host's identities, to ensure fairness of service provision. The service host reports a malicious mobile agent to a third party authority who is able to track down the corresponding agenthost and penalize it accordingly. The differences between our protocol and Lin's is that our protocol is designed for a mobile agent to sign documents autonomously on behalf of its owner with necessary security algorithms and measures, whereas Lin's protocol does not specify the type of service, *i.e.* task, the agent willperform. In addition, Lin's protocol requires that the agent host obtains a certified permission from the service host to provide the service it needs prior to dispatching the agent to ensure fairness of service provision. In our protocol, there is no pre-transaction communication be

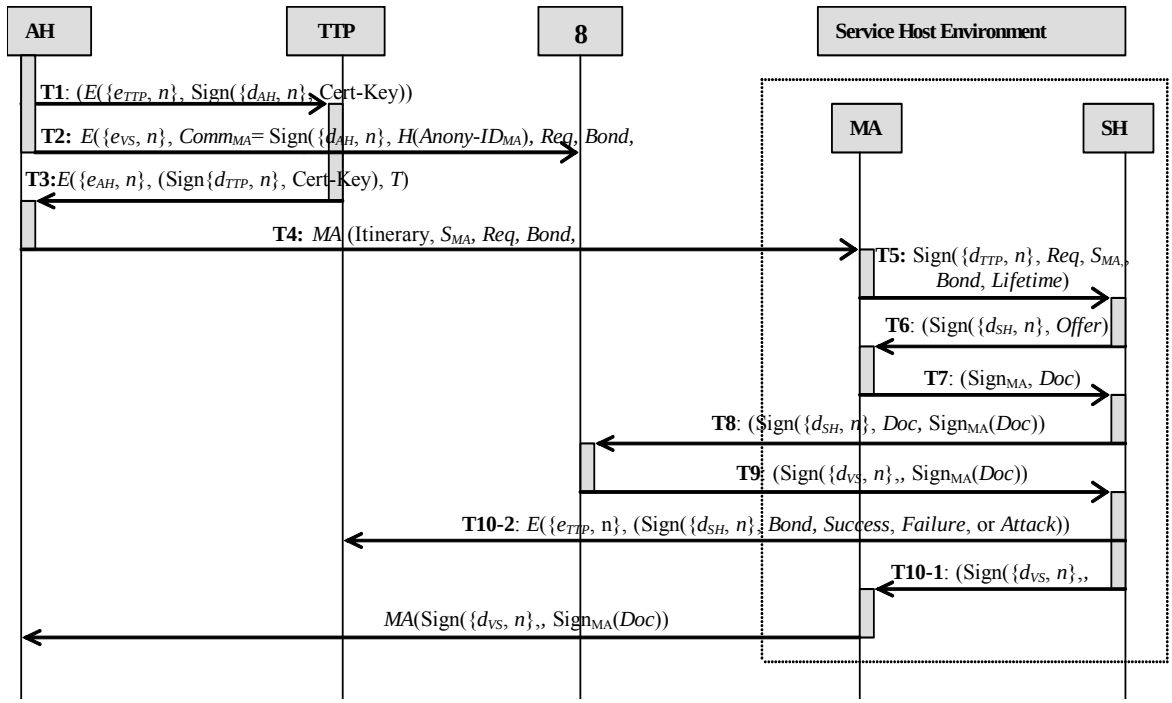


Figure 3. Collusion-Resistance Distributed Agent-Based Signature Delegation (CDASD) Protocol.

Table 3. Summary of the methods/algorithms performed by each party with their inputs and outputs.

Party	Methods/Algorithm	Input	Output
AH	Signature Key Generation method – 1&2	$Anony\text{-}ID_{MA}$, r, e_{TTP}	Z
	Chaum's Blind Signature method - Unblinding	T	S_{MA}
TTP	Chaum's Blind Signature method - Blinding	Z, d_{TTP}	T
MA	Agent Signature Generation method	$Doc = (Offer, Req)$, S_{MA}	$Sign_{MA}(Doc)$
VS	Agent Signature Verification method-2	(Sign($\{d_{SH}, n\}$, Doc), $\text{Sign}_{MA}(\text{Doc})$, $H(Anony\text{-}ID_{MA})$, e_{TTP}	Trure/False

tween the agent host and the service host, the **Table 4** shows the computational overhead measured in terms of the total number of encryption, decryption, signing signatures, and verifying signatures operations performed at each protocol phase. In order to compare the two protocols in terms of these operations, a measuring unit is needed to perform the comparison quantitatively. This unit is chosen to be the exponentiation operation as it is considered the most resource-consuming operation. Assuming that RSA cryptosystem is used by both protocols, each of the above mentioned operations includes one exponentiation operation.

From **Table 4**, it can be seen that the CDASD protocol enjoys a saving of approximately 52% in the number of exponentiation operations (considered as the most resource-consuming operations), comparing with Lin's protocol. This savings are mainly due to the fact that in Lin's protocol, all the messages exchanged between the protocol's parties are signed and encrypted. Whereas, in our protocol, only the messages that contain security-sensitive information are signed and encrypted. For example, message **T1** sent from *AH* to the *TTP* is en crypted because it contains the blinded signature key and a transaction related information to protect it from being eavesdropped and misused by an outsider attacker.

In addition to the computational savings, the CDASD protocol provides extra service that Lin's protocol does not. These services are:

- The ability of the mobile agent to generate digital signature autonomously and anonymously on behalf of its owner. This service is very important in facilitating e-commerce application. The CDASD protocol ensures the security requirements for this service, *i.e.* verifiability and unforegability of the signature together with non-repudiation of signature origin and receipt (as explained in the next section).
- Collusion-resistant property in which the *VS* and the service host, if collude together, should find it difficult to get any advantage over a mobile agent and the agent host (as explained in the next section).

5.2. Security Analysis

In this section, we analyze the security properties of the CDASD protocol demonstrating that it satisfies all the security requirements stated in Subsection 3.1.1.

S1) Verifiability of the Signature:

VS is able to verify the validity of mobile agent's signature $\text{Sign}_{MA}(Doc)$ using the information included in commitment $Comm_{MA}$, which is generated and sent by *AH*. It is worth noting that *VS* is able to verify $\text{Sign}_{MA}(Doc)$ without accessing *MA*'s anonymous identity $Anony\_ID_{MA}$. This feature supports the anonymity property of *MA* and, in turn, the fairness of service provision (next).

S2) Unforgeability of the Signature:

Since the mobile agent signature key S_{MA} is derived from the agent anonymous ID ($Anony\text{-}ID_{MA}$) that is only known to *AH* and is a one-time, it would be difficult for another party to forge the signature key without know-

Table 4. Computation Overhead.

		Encryption	Decryption	Signature	Verification of signatures
Lin's protocol	Phase 1: service registration and inquiry	5	4	5	5
	Phase 2 : applying for services' permissions	2	2	2	2
	Phase 3: requiring *SH*'s services	7	7	2	2
Total		**14**	**13**	**9**	**9**
CDASD protocol	Phase 1: certified signature key acquisition	4	2	3	2
	Phase 2: service request & signature generation	0	0	2	2
	Phase 3: signature verification	1	1	2	3
Total		**5**	**3**	**7**	**7**

ledge of *Anony_ID*$_{MA}$. However, there are two scenarios where the signature might be forged:

- *VS* receives the hash of *Anony_ID*$_{MA}$, *i.e.* *H*(*Anony-ID*$_{MA}$) in *Comm*$_{MA}$ sent by *AH*. *VS* might try to use this hash to compute S_{MA}. For doing so, *VS* needs also to know the private key of the *TTP*, *i.e.* e_{TTP}, which only *TTP* has knowledge of. Therefore, it is difficult for *VS* to re-generate S_{MA} and forge *MA*'s signature.
- In *TTP*: the *TTP* certifies the signature key by calculating $T = H(Anony - ID_{MA})^{d_{TTP}} \times r$. For the *TTP* to obtain the signature key $S_{MA} = H(Anony - ID_{MA})^{d_{TTP}}$ from *T*, it has to know the blinding factor *r*, which only *AH* has knowledge of. It is also difficult for the *TTP* to obtain S_{MA} from *T* due to the difficulty of factoring large primes, *i.e.* factoring *T* to get S_{MA} and *r*

S3) Non-Repudiation of Signature Origin:

This security property is achieved in our protocol by the following measures:

- The verification Verification VS-1 performed by the *VS* ensures that the signature Sign$_{MA}$(*Doc*) is generated by using a signature key that is generated by *AH*. This is because *VS* uses *H*(*Anony-ID*$_{MA}$) received in *Comm*$_{MA}$, which is signed by *AH*. Therefore, *AH* cannot deny the fact that it has generated the signature key.
- If *AH* denies signing *Doc* after a successful completion of the transaction, *i.e.* *AH* has received the signed document certified by *VS* (Sign{d_{VS}, n}, Sign$_{MA}$(*Doc*)), *SH* can then send a complain to the *TTP*. The complain contains the certified signature (Sign{d_{VS}, n}, Sign$_{MA}$(*Doc*)) together with the *Bond* value. The *TTP* wll then fetch *Bond* value in AH-MA Relation table and ob- tain the corresponding *AH*'s identity (*ID*$_{AH}$). *TTP* will then sign both *ID*$_{AH}$ and *Bond* and send it to *SH* as a proof of holding *AH* responsible for generating the signature.

S4) Non-Repudiation of Signature Receipt:

This requirement is achieved through the use of the certified signature (Sign{d_{VS}, n}, Sign$_{MA}$(*Doc*)) signed by the *VS* and sent to *AH* through *MA* in **T10-1**. As *SH* cannot verify the mobile agent signature, *SH* has to send a signature verification request to *VS* in **T8**, which proves that *SH* has actually received *MA*'s signature on *Doc* if the verification Verification VS-1 outcome is positive and *VS*'s signature on Sign$_{MA}$(*Doc*) is produced. Therefore, *SH* cannot deny later that it has received *MA*'s (representing *AH*) signature on *Doc*.

S5) Collusion-Resistance:

In order to check if the CDASD protocol satisfies this requirement, we first have to look at the data items both *VS* and *SH* have or know, shown in **Table 5**.

From **Table 5**, it can be seen that the piece of data owned by *VS* and of an interest to *SH* is *H*(*Anony-ID*$_{MA}$). *SH* might use this information to guess *MA*'s or *AH*'s identities, hence, violate the anonymity properties and, in turn, the fairness of service provision property. This attack is thwarted in our protocol as follows. As mobile agent's anonymous identity (*Anony-ID*$_{MA}$) is randomly generated and hashed, it does not reveal any information about either *MA*'s or *AH*'s identities. Furthermore, *Anony-ID*$_{MA}$ is freshly generated for each transaction, *i.e.* one-time only, which means *SH* will find it difficult to use this information to link together two different transactions performed by the same agent in a hope of guessing *MA*'s identity. Therefore, it is difficult for the *VS* and the *SH*, if collude together, to get any advantage over the *MA* and the *AH*.

S6) Unlinkability:

By looking at the contents of the signature (Sign$_{MA}$(*Doc*) = ($H(Anony - ID_{MA})^{d_{TTP} \times H(Doc)} \bmod n$, *Doc*), generated by *MA* using the signature key S_{MA}, it can be seen that the signature does not have any information that can be used to link it with other signature generated by the same mobile agent *MA* (in a different transaction) or the same agent host *AH*. This is because the mobile agent signature key S_{MA} is computed using a freshly generated mobile agent anonymous identity (*Anony-ID*$_{MA}$). This key is one-time only, hence, is used to generate one signature on the document for one transaction. This unlinkability feature supports the anonymity property to be discussed next.

S7) Anonymity:

In addition to the anonymity provided by the unlinkability of different signatures to the same *MA* or *AH*, anonymity is also provided through the contents of the mobile agent itself. That is, the data the mobile agent carries while roaming the network, *i.e.* (*Itinerary*, S_{MA},

Table 5. Data items owned/known by *VS* and *SH*.

	VS	SH
Knows/Have	• Contents of Verification Information table: (*H*(*Anony-ID*$_{MA}$, *Req*, *Lifetime*, Sign$_{MA}$(*Doc*), Sign({d_{SH}, n}, *Doc*), *Verification Result*) • *Comm*$_{MA}$ = Sign({d_{AH}, n}, *H*(*Anony-ID*$_{MA}$), *Req*) • $Sign(\{d_{SH}, n\}, Doc), Sign_{MA}(Doc)$ • *TTP*'s and *SH*'s public keys • Sign({d_{VS}, n}, Sign$_{MA}$(*Doc*)	• *MA*'s contents: (S_{MA}, *itinerary*, *Bond*, Req) • *TTP*'s public key • Sign$_{MA}$(*Doc*) = (*Doc*, *D*) • Sign({d_{SH}, n}, *Doc*) = $H(Doc)^{d_{SH}} \bmod n$ • Sign({d_{SH}, n}, *Doc*) = $H(Doc)^{d_{SH}} \bmod n$ • Sign({d_{SH}, n}, *Offer*). • Sign({d_{VS}, n}, Sign$_{MA}$(*Doc*)

Bond, *Lifetime*) neither reveals *MA*'s nor *AH*'s identities. Therefore, it will be difficult for *SH*s visited by the agent to obtain any information that leads to the agent's source. One may argue that if the agent does not carry any information regarding it's, or its source's, identity, then how can *SH*s authenticate this agent? The agent is authenticated through the *TTP*'s signature on its contents, which is sent to *SH* in **T5**. Verification SH-1, performed by *SH*, verifies *TTP*'s signature on the agent's contents, hence, authenticate the agent by the trust *SH* hold for the *TTP*. This trust stems from the fact that the *TTP* only certifies agents of whom their owners are trustworthy, *i.e.* not blacklisted.

S8) Fairness of Service Provision:

Due to the unlinkability and anonymity properties, the *SH* will not have a mean to distinguish between the authenticated mobile agents as to which to provide its service first. Therefore, the *SH* will serve all the agents on the first-come-first-served basis, hence, fairness of service provision is satisfied.

S9) Agent Host Accountability:

As the mobile agent in our protocol is anonymous, *i.e.* untraceable, one may question about the ability of *SH*s, if attacked by malicious agents, to get hold of them. In our protocol, the *TTP*, in collaboration with the *SH*, is able to detect and penalize, *i.e.* blacklist, an agent host when its agent acts maliciously. In details, an *AH* generates a *Bond* for each *MA* and pass it to the *MA* before being dispatched to perform its task. This *Bond* is sent to the *TTP* to record it with other related information in AH-MA Relation table. When the *MA* enquires about a service in *SH*, it submits its *Bond* value to that *SH*. If an *MA* attacks an *SH*, the *SH* will send an 'Attack' flag together with the *Bond* value as an outcome of the transaction in message **T10-2** to the *TTP*. The *TTP* compares the *Bond* with the value in the Bond field of the AH-MA Relation table to fetch the identity of the *AH* who sent this *MA* and penalize accordingly.

6. Conclusions and Future Work

6.1. Conclusions

This research has addressed the mobile agent-based anonymous signature delegation issue by critically analyzing related works and highlighting their shortcomings. We then presented a novel Collusion-Resistant Distributed Agent-Based Signature Delegation (CDASD) protocol, which incorporates three methods as its building blocks, namely, Agent Signature Key Generation method, Agent Signature Generation method, Agent Signature Verification method. The protocol enjoys the following features. Firstly, it makes use of Chaum's blind signature scheme to allow for a trusted third party (TTP) to blindly certify the mobile agent signature key for mobile agent's anonymity. The mobile agent, while residing at the service host, does not reveal any information about its host's identity, which deprive the service host from favoring a mobile agent on the others, hence, ensuring fairness of service provision. Secondly, the protocol presents a mechanism to track down malicious mobile agents and penalize their hosts accordingly. Thirdly, the protocol provides non-repudiation of signature generation and receipt so that neither the mobile agent and its host nor the service host can deny generating and receiving the signature, respectively. The protocol introduces a verification server to verify the signature generated by the mobile agent in such a way that even if colluding with the service host, both parties will not get more information than what they already have. Security-sensitive messages exchanged between protocol's parties are signed and encrypted to prevent unauthorized disclosure or tampering with the contents of these messages. The protocol analysis shows that, in addition of fulfilling the security requirements specified in Subsection 3.1.1, it is more efficient in comparison with the related work. Our protocol can be applied to many applications, e.g., e-/m-commerce, grid computing, and ubiquitous computing due to the properties of mobile agents.

6.2. Future Work

We have the following recommendations for future work:

- Formal verifications of the security properties of the protocol using a verification tool, *i.e.* the Alternating Temporal Logic and the model checker MOCHA [24,25].
- Implementation of the protocol using Grasshopper mobile agent framework [26] and Java libraries and evaluation of the implemented protocol's performance.
- Our solutions have only addressed the problem of securing mobile agent-based e-commerce transactions. As we mentioned earlier, mobile agents can also be used in Grid computing environment. The security issues in the context and solutions to problems in the Grid Security Infrastructure are venues for further research.
- Mobile agents have an important role to play in facilitating m-commerce applications where customers, *i.e.* agent owners, use resource-limited devices such as PDAs, pocket PCs and mobile phones to perform m-commerce transactions. Care has already been taken in the design of our protocol to minimize the computational costs. Further research into how to integrate our protocol with the existing wireless technologies, e.g. UMTS, is needed for the mobile agent-based m-commerce applications to achieve their full potential.

7. Acknowledgment

This research was supported by King Abdulaziz University, Jeddah, Saudi Arabia, under Grant 427/515.

8. References

[1] A. Asokan, V. Shoup and M. Waidner, "Optimistic Fair Exchange of Digital Signatures," *IEEE Journal on Selected Areas in Communication*, Vol. 18, 2000, pp. 591-610.

[2] O. Bamasak, "Delegating Signing Power to Mobile Agents: Algorithms and Protocol Design," PhD Thesis, School of Computer Science, the University of Manchester, UK, 2006.

[3] F. Bao, R. H. Deng and W. Mao, "Efficient and Practical Fair Exchange Protocols with Off-Line TTP," *Proceedings of the IEEE Symposium on Security and Privacy*, 1998, pp. 77-85.

[4] M. Blum, "How to Exchange (Secret) Keys," *ACM Transactions on Computer Systems*, Vol. 1, No. 2, 1983, pp. 175-193.

[5] C. Boyd and E. Foo, "Off-Line Fair Payment Protocols Using Convertible Signature," *Advances in Cryptology -Proceedings of Asiacrypt*'98, *Lecture Notes in Computer Science* 1514, 1998, pp. 271- 285.

[6] L. Chen, "Efficient Fair Exchange with Verifiable Confirmation of Signatures," *Advances in Cryptology-Proceedings of Asiacrypt*'98, *Lecture Notes in Computer Science* 1514, 1998, pp. 286-299.

[7] R. H. Deng, L. Gong, A. A. Lazar and W. Wang, "Practical Protocol for Certified Electronic Mail," *Journal of Network and System Management*, Vol. 4, No. 3, 1996, pp. 279-297.

[8] S. Even, O. Golreich and A. Lempel, "Randomized Protocol for Signing Contracts," *Communications of the ACM*, Vol. 28, No. 6, 1985, pp. 637-647.

[9] M. K. Franklin and M. K. Reiter, "Verifiable Signature Sharing," *Advances in Cryptology-Proceedings of Eurocrypt*'95, *Lecture Notes in Computer Science* 921, 1995, pp. 50-63.

[10] J. A. Garay, M. Jakobsson and P. MacKenzie, "Abuse-Free Optimistic Contract Signing," *Advances in Cryptology-Proceedings of Crypto*'99, *Lecture Notes in Computer Science* 1666, 1999, pp. 449-466.

[11] M. Jakobsson, K. Sako and R. Impagliazzo, "Designated Verifier Proofs and their Applications," *Advances in Cryptology-Proceedings of Eurocrypt*'96, *Lecture Notes in Computer Science* 1070, 1996.

[12] T. Okamoto and K. Ohta, "How to Simultaneously Exchange Secrets by General Assumptions," *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, 1994, pp. 184-192.

[13] C. Wang and C. Yin, "Practical Implementations of a Non-disclosure Fair Contract Signing Protocol," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science*, Vol. E89-A, No. 1, 2006, pp. 297-309.

[14] J. Zhou and D. Gollmann, "A Fair Non-Repudiation Protocol," *Proceedings of* 1996 *IEEE Symposium on Security and Privacy*, 1996, pp. 55-61.

[15] J. Zhou and D. Gollmann, "An Efficient Non-Repudiation Protocol," *Proceedings of* 1997 *IEEE Computer Security Foundations Workshop* (*CSFW* 10), 1997, pp. 126-132.

[16] M. Lin, C. Chang and Y. Chen, "A Fair and Secure Mobile Agent Environment Based on Blind Signature and Proxy Host," *Computers & Security*, Vol. 23, 2004, pp. 199-212.

[17] D. Chaum, "Blind Signatures for Untraceable Payments," *Proceedings of CRYPTO*'82, 1983, pp. 199-203.

[18] J. Kim, G. Kim and Y. Eom, "Design of the Mobile Agent Anonymity Framework in Ubiquitous Computing Environments," *IEICE Transactions on Information and Systems*, Vol. E89-D, No. 12, 2006, pp. 2990-2993.

[19] R. L. Rivest, A. Shamir and L. M. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," *Communication of Association for Computing Machinery*, Vol. 21, No. 2, 1978, pp. 120-126.

[20] National Institute of Standard and Technology (NIST), Secure Hash Standard, Federal Information Processing Standards Publication (FIPS 180-1).

[21] M. Fowler, "UML Distilled: A Brief Guide to the Standard Object Modeling Language," 3rd Edition, The Addison-Wesley Professional, 2003.

[22] U. Wilhelm, "Cryptographically Protected Objects," Technical Report, Ecole Polytechnique Federale de Lausanne, Switzerland, 1997.

[23] F. Hohl, "Time Limited Blackbox Security: Protecting Mobile Agents from malicious Hosts," *Mobile Agents and Security, Lecture Notes in Computer Science*, Vol. 1419, 1998, pp. 92-113.

[24] S. Kremer and J. Raskin, "A Game-Based Verification of Non-Repudiation and Fair Exchange Protocols," *Proceedings of the 12th International Conference on Concurrency Theory, Lecture Notes in Computer Science*, Vol. 2154, 2001, pp. 551-566.

[25] S. Kremer and J. Raskin, "Game Analysis of Abuse-Free Contract Signing," *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, 2002, pp. 206-220.

[26] "Grasshopper Mobile Agent Platform." http://www.grasshopper.de

[27] M. Mambo, K. Usuda, and E. Okamoto, "Proxy Signatures for Delegating Signing Operation," *Proceedings of the 3rd ACM Conference on Computers and Communications Security*, 1996, pp. 48-57.

[28] N. Borselius, C. Mitchell and A. Wilson, "A Pragmatic Alternative to Undetachable Signatures," *ACM SIGOPS Operating Systems Review*, Vol. 36, No. 2, 2002, pp. 6-11.

[29] A. Romao and M. Silva, "Secure Mobile Agent Digital Signatures with Proxy Certificates," *E-Commerce Agents, Marketplace Solutions, Security Issues, and Supply and demand, Lecture Notes in Computer Science*, Vol. 2033,

2001, pp. 206-220.

[30] B. Lee, H. Kim, J. Baek and K. Kim, "Secure Mobile Agent Using Strong Non-designated Proxy Signature," *Proceedings of the 6th Australian Conference on Information Security and Privacy*, *Lecture Notes in Computer Science*, Vol. 2119, 2001, pp. 474-486.

[31] S. Kim, S. Park and D. Won, "Proxy Signatures, Revisited," *Proceedings of the International Conference on Information and Communications Security*, *Lecture Notes in Computer Science*, Vol. 1334, 1997, pp. 223-232.

[32] K. Zhang, "Threshold Proxy Signature Schemes," *Proceedings of Information Security Workshop, Lecture Notes in Computer Science*, Vol. 1396, 1997, pp. 282-290.

[33] H. Kim, J. Baek, B. Lee and K. Kim, "Secret Computation with Secrets for Mobile Agent Using One-Time Proxy Signature," *Proceedings of the* 2001 *Symposium on Cryptography and Information Security*, 2001, pp. 845-850.

[34] K. Shum and V. Wei, "A Strong Proxy Signature Scheme With Proxy Signer Privacy Protection," *Proceedings of the* 11*th IEEE International Workshop on Enabling Technologies*: *Infrastructure for Collaborative Enterprises*, 2002, pp. 55-56.

[35] J. Herranz and G. Saez, "Fully Distributed Proxy Signature Schemes," *Cryptology ePrint Archive*, 2002. http://eprint.iacr.org/2002/051

[36] H. Wang and J. Pieprzyk, "Efficient One-Time Proxy Signatures," *Proceedings of ASIACRYPT, Lecture Notes in Computer Science*, Vol. 2894, 2003, pp. 507-522.

[37] Y. Yong, C. Xu, X. Huang and Y. Mu, "An Efficient Anonymous Proxy Signature Scheme with Provable Security," *Computer Standards & Interfaces*, Vol. 31, No. 2, 2009, pp. 348-353.

[38] Z. Shao, "Provably Secure Proxy-Protected Signature Schemes Based on RSA," *Computers and Electrical Engineering*, Vol. 35, No. 3, 2009, pp. 497-505.

