

Reputation management and signature delegation: A distributed approach

Omaima Bamasak · Ning Zhang

© Springer Science + Business Media, LLC 2006

Abstract In this paper, we present a novel protocol, called Distributed Signcryption with Verifiable Partial Signature (DiSigncryption) protocol, to allow an agent owner to securely distribute his signing capability among a set of trusted third party hosts (*TTP-hosts*) via a mobile agent. The protocol incorporates three schemes: a novel Distributed Reputation Management scheme, a modified version of the Distributed Signcryption method proposed in [23], and an extended version of the Agent-based Threshold Proxy Signcryption (ATPS) protocol proposed in [2]. The security properties of the proposed protocol are analyzed, and the protocol is compared with the most related work.

Keywords Proxy signature · Signature delegation · Mobile agent-based e-commerce · Reputation management

1. Introduction

Mobile agent paradigm can bring many benefits to distributed systems, such as great flexibility and improved performance due to its properties of mobility and autonomy. As a result, mobile agents are believed to be playing an important role in future electronic/mobile commerce (e-/m-commerce) systems. One of the most sensitive tasks a mobile agent is expected to perform in a remote host while participating in an e-/m-commerce transaction is signing a digital signature on behalf of its owner for the purpose of transaction authentication. In other words, a mobile agent may act as a proxy signer and produce signatures on behalf of the original signer

O. Bamasak (✉) · N. Zhang
School of Computer Science, The University of Manchester, Oxford Road, Manchester, M13 9PL,
United Kingdom
e-mail: obamasak@cs.man.ac.uk

N. Zhang
e-mail: nzhang@cs.man.ac.uk

(i.e. the agent owner) autonomously on a remote host. Such signatures are referred to as proxy signatures in the literature. The remote host may not be trustworthy. We here use an example to illustrate the security threats imposed by a malicious remote host against the mobile agents executed on the host. Say Alice needs to pay an overseas visit to her business partner in Germany, and would like to search for and purchase an airline ticket on the web. Since the search may take a while and Alice cannot afford to spend the time to do the search in person, she delegates this task to a mobile agent. Alice may specify some conditions such as if the price of a ticket is below a certain threshold, the agent should book the flight at once on her behalf. It is clear that, in this scenario, a remote airline host has the incentive to misbehave or to manipulate the agent. For example, the host may force the agent to sign a deal that is above the specified threshold, or the host may forge the agent signature(s) altogether by spying on the signature key carried by the agent. In such circumstances the agent owner will be liable for the deal and held responsible for the forged signatures. Previous research works [15, 16, 18, 21, 22, 29] have proposed some solutions to this problem, i.e. how to secure a signature key carried by a mobile agent. However, these solutions have mainly focused on the protection of the signature key against third party perpetrators, and are weak in tackling threats imposed by the other side of the business deal, i.e. the merchant host. For example, the work by [16, 21, 22, 29] has failed to provide non-repudiation of signature receipt service leaving room for the merchant to later deny that a booking has been made. The work by Lee et al. [18] does not protect the signature key (also called proxy key) from being misused by the remote (merchant) host. Though Kim [15] has recognized and addressed some of these weaknesses, but the solution proposed is computationally expensive.

Recently, Bamasak and Zhang [1, 2] have proposed a more secure and efficient protocol for signature delegation to a mobile agent. This scheme makes use of a Trusted Third Party (TTP), played by a single trusted host, to assist the mobile agent in signature generation and in transactional evidence management for the provision of non-repudiation of signature receipt service. It is clear that this centralized TTP approach creates a performance/reliability bottleneck and introduces a single point of failure for the protocol. Moreover, due to its size and openness, it is increasingly difficult to protect any single system against the sort of attacks proliferating on the Internet today.

One established method for enhancing the fault tolerance of the TTP is to distribute its functionality among a set of trusted hosts rather than relying on a single host. In this way, the robustness of the protocol can be strengthened as, to compromise the TTP, a majority of, if not all of, the trusted hosts would have to be compromised. Then, an important question arises at this stage is: how should the agent owner decide on the group of hosts that are most trusted to perform the role of the TTP? In other words, what should be the criteria on which the agent owner makes the decision as whether or not a given host should be selected to join the set that plays the role of the TTP collectively? Obviously, the agent owner should select the *TTP-hosts* that have an acceptable level of reputation. Conventional security solutions and cryptographic methods, such as the use of passwords and digital certificates, alone are not sufficient for us to evaluate the trustworthiness of a *TTP-host*. They can help us to establish whether a party is authenticated and authorized to take certain actions. They cannot guarantee that the party will perform as promised and deliver a trusted service. In other

words, conventional cryptographic solutions alone cannot hold a party accountable for its actions and cannot address the problem of reputation.

This paper addresses the issue of reputation and proposes a solution named the DiSigncryption protocol, which provides a secure and efficient approach to mobile agent-based signature delegation and overcomes the drawback of the ATPS protocol—the use of a single *TTP-host*. The major contributions of the paper are as follows:

- (1) It has specified security and performance requirements for distributed agent-based signature delegation.
- (2) It has proposed a novel distributed reputation management scheme suited to the signature delegation model proposed in [2]. This scheme allows an agent owner to assign and update trust and reliability values for each *TTP-host* that the agent owner has dealt with. These values reflect a credit level for the *TTP-host* over time, and the credit level may increase or decrease depending on the behaviour of the *TTP-host* concerned.
- (3) It has presented eight cryptographic primitives that are used in the design of the DiSigncryption protocol. They are, namely, a Group Public Key Generation method, a Proxy Key Generation method, a Proxy Key Shares Generation method, a Distributed Signcryption scheme, a Partial Proxy Signature Generation method, a Partial Proxy Signature Verification method, a Complete Proxy Signature Generation method, and a Proxy Signature Verification method. The Group public Key Generation method, which is proposed in [23], has been used to generate a group public key and a set of secret keys shared between the agent owner and the participating *TTP-hosts*. The Proxy Key Generation method, the Proxy Key Shares Generation method, the Complete Proxy Signature Generation method, and the Proxy Signature Verification method have been used in the design of the ATPS protocol in [2] to facilitate the generation of a proxy key, the generation of proxy key shares, the reconstruction of the complete proxy signature from the signature shares, and the verification of the reconstructed proxy signature, respectively. The rest of the cryptographic primitives are novel, i.e. newly proposed in this paper and they are:

- The Distributed Signcryption scheme for a secure delivery of each proxy key share to its corresponding *TTP-host*.
- The Partial Proxy Signature Generation method for the generation of partial signatures and their corresponding commitments.
- The Partial Proxy Signature Verification method for the verification of the partial proxy signature through the use of the commitment generated by the Partial Proxy Signature Generation method.

The above mentioned eight cryptographic primitives are integrated to form an extended version of the ATPS protocol to address the issue of distributed signature delegation.

- (4) The newly designed Distributed Reputation Management scheme is integrated into the extended version of the ATPS protocol, resulting in a novel Distributed Signcryption with Verifiable Partial Signature (DiSigncryption) protocol
- (5) The DiSigncryption protocol has been analyzed with regard to its performance measured in terms of communication overhead (i.e. the number and sizes of

protocol messages) and computational overhead (i.e. the number of multiplication and exponentiation operations used in the protocol design). A security analysis has also been conducted to demonstrate that the DiSigncryption protocol satisfies its security requirements. The analysis has been compared with the work related most to ours showing that our protocol is more secure and efficient. This implies that our protocol is suited to agent-based m-commerce applications for which the network is characterized by low and/or expensive bandwidth and the mobile devices are expected to have thin computational and storage capabilities.

The proposed distributed reputation management scheme and the DiSigncryption protocol can be applied to a wide range of distributed applications, for example:

- E-/m-commerce applications:
 - Contract signing where the agent owner and the remote host represent the contract partners. Upon transaction completion, each partner gets the contract signed by all the involved partners.
 - E-purchase where the agent owner represents a customer who wants to purchase a specified good from a remote host representing a merchant. By the end of the transaction, the merchant gets a payment order signed by the customer and the customer either gets the good or a proof of payment recipient signed by the merchant.
- Grid computing (resource sharing): the agent owner represents a client who wants to coordinate his/her job over a computational grid of computers. A mobile agent autonomously navigates over the Internet and coordinates task executions mining available resources. It migrates from one computer, i.e. remote host, to another whenever the current resource becomes unavailable. The mobile agent, upon completing a task on a remote host, provides the remote host with his owner's signature on a summary of that task execution for account purpose. The remote host also provides his signature to the mobile agent for the same reason.
- Trust management in distributed applications: the Distributed Trust Management scheme can be applied in distributed applications where a group of remote hosts perform a trusted service for a client. This scheme enables the client to control and manage the trust placed in each of the remote hosts.

The remainder of the paper is organized as follows. Section 2 summarizes the related work in distributed trusted services and reputation management systems. Section 3 outlines the security and performance requirements, and the principles used, for the design of the DiSigncryption protocol. Section 4 details the Distributed Trust Management scheme. Section 5 presents the DiSigncryption protocol. In Section 6, the protocol is analyzed against the requirements specified and compared with related work, and finally, our conclusions and future work are given in Section 7.

2. Related work

As our work integrates a reputation management system into distributed trusted services, existing works in both of these areas are reviewed and analyzed in this section.

2.1. Distributed trusted services

Distributed approach to the provision of a trusted service by a set of multiple hosts has attracted fair amount of research interests in the last decade [3–5, 9, 11, 14, 17, 19, 23, 28, 32, 33, 38]. The solutions proposed can be classified into two categories. In the first category [5, 9], the set of hosts is managed by a single trusted entity that plays the role of a group manager responsible for performing some security sensitive tasks and distributing operational tasks among the group members (i.e. *TTP-host*). In these solutions, the group manager is still a single point of operational and security failure to the system, so the solutions are not really distributed. In the second category [3, 9, 11, 14, 17, 19, 28, 32, 33, 38], all group members communicate and collaboratively perform the tasks. The downside of this approach is that it introduces communication overhead and additional processing load among the group members.

2.2. Reputation management systems

A number of trust/reputation management systems and mechanisms have been proposed in e-commerce context [13, 20, 35, 37]. The authors in [37] have developed a collaborative reputation mechanism allowing personalized evaluations for the ratings assigned to various parties for the prediction of their reliabilities in e-commerce context. However, the work does not clearly identify the metrics, i.e. criteria, upon which the rating, i.e. reputation value, for each party is updated. The methods presented in [13, 20, 35] only deal with peer-to-peer trust. They do not provide a solution for the selection of a group of *TTP-hosts* that can jointly perform a security-sensitive task.

In Internet-based electronic marketplaces, there are a few working on-line reputation systems such as the feedback system used by eBay [6] and Yahoo!Auction [36]. These systems allow participants of a transaction to rate each other with a value of +1 for positive feedback, 0 for neutral, and -1 for negative feedback. Only winning bidders and sellers may submit feedback for their completed transactions. Transaction participants may also fill in a written response. The reputation value of a participant is then calculated as the sum of all ratings this participant has received since s/he started dealing in these on-line marketplaces. Any user with a reputation value of -4 or less would be suspended from dealings on eBay. This approach is linear and single-factor based (i.e. positive or negative feedback), and often fails to capture the trustworthiness of parties effectively. For example, a user who has 100 positive feedback responses will have the same aggregated feedback value as a user who has had 300 positive and 200 negative responses.

The reputation management scheme proposed in this paper has addressed the weaknesses identified above. Namely, the scheme has defined the criteria and an algorithm for the selection of a set of most reputable *TTP-hosts* to which a security-sensitive task will be delegated by the agent owner. Secondly, it has defined the criteria and an algorithm by which the agent owner updates the reputation value associated with each of the *TTP-hosts* according to the feedbacks received from a party, e.g. the merchant host, that has interacted with the *TTP-hosts* to jointly perform the transaction. This scheme is presented in details in Section 4 next.

3. Requirements specification

Prior to presenting our novel reputation management scheme and the DiSyncryption protocol, here in this section, we first specify security and functional requirements which the protocol is aimed at satisfying and then outline the principles used for the protocol design.

3.1. Security requirements

- (S1) *Proxy key confidentiality*: The proxy key should not be disclosed to any entity other than the agent owner, i.e. only the agent owner should have a complete knowledge of the proxy key carried by an agent.
- (S2) *Partial proxy key share confidentiality*: A proxy key share sh_i should only be revealed to the designated proxy entity *TTP-host_i*.
- (S3) *Proxy signature unforgeability*: A valid proxy signature can only be cooperatively generated by the delegated by at least F (out of N) proxy signers. Non-delegated proxy signers should have no capability to generate a valid proxy signature. Also, $(F-1)$ or less proxy signers have no capability of forging a valid proxy signature.
- (S4) *Partial proxy signature verifiability*: Partial proxy signatures should be verifiable, i.e. the signature verifier should be able to verify the validity of a partial proxy signature through the use of some parameters.
- (S5) *Non-repudiation of signature origin*: It should be difficult for the original signer (i.e. the agent owner) of a proxy signature to falsely deny that it has generated the proxy signature (via a mobile agent).
- (S6) *Non-repudiation of signature receipt*: It should be difficult for a signature recipient to falsely deny that it has received the proxy signature, if this signature is taken as the proof of a deal agreed between the proxy signer (i.e. the agent) and the recipient.
- (S7) *Fairness*: This requirement indicates that, once a deal is agreed, then either the original signer and the signature recipient have both received the proxy signature on the deal, or neither of them has received anything useful. In other words, if the signature recipient has received the proxy signature of the original signer on a deal M , the original signer should have also received the signature recipient's signature on M , and vice versa.
- (S8) *Restricting the misuse of proxy key*: Measures have been taken so that it would be difficult for any entity to misuse a proxy key for purposes other than that originally specified by the original signer.
- (S9) *Confidentiality of the document to be signed*: Only the parties signing the document should have a complete knowledge of the document to be signed. In other words, no party other than the agent owner, the remote merchant host, and the *TTP-hosts* can access to the contents of the document to be signed.
- (S10) *TTP-hosts accountability*: Any misbehavior by any of the *TTP-hosts* should be detectable and accounted for.

3.2. 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, especially for the agent owner side that is typically a mobile device with limited resources.
- (P2) *Protocol robustness*: The protocol allows multiple *TTP-hosts* to play the role of a TTP achieving robustness and fault-tolerance.
- (P3) *Dynamic TTP-host selection*: The number of *TTP-hosts* used for the provision of trusted services supporting our DiSigncryption protocol is determined on per-transaction basis depending on the value of the transaction and the reputation of the *TTP-hosts* to be used. This requirement implies the use of a non-linear reputation model achieving cost-effective approach to security and reliability.

3.3. Design principles

To satisfy the requirements specified above, the designs of our reputation management scheme, the DiSigncryption protocol, and the trust model has taken into account a number of measures. These measures are described below together with their justifications.

Measure 1. In our distributed reputation model, a *TTP-host's* reputation is measured in terms of a trust level and a reliability level, both of which are aggregated over a specified past period. The trust level reflects the truthfulness of the *TTP-host* in executing a transaction and the reliability level reflects its robustness in providing the TTP service. Both levels are functions of the following parameters:

- (1) *Transaction outcome feedback*. After each transaction is completed, the merchant host assigns a feedback measured in terms of trust and reliability values to each participating *TTP-host*. The values given are dependent on the outcome of the transaction performed by the *TTP-host*. For example, if the transaction outcome is positive, the trust value associated with the *TTP-host* will be 'Yes', and if the outcome is negative, then the trust value will be 'No'. For the reliability value, if the merchant host did not receive an expected message from the *TTP-host* after a certain period of time, the merchant will assign 'No' to the reliability value associated with the *TTP-host*, otherwise, the reliability value will be 'Yes'. This per-transaction trust value reflects how well this *TTP-host* has fulfilled its part of the protocol execution in performing this transaction, and the per-transaction reliability value reflects how satisfied the merchant is with regard to the quality of the information or service provided by the *TTP-host* in this transaction. The overall trust and reliability values owned by a *TTP-host* is an aggregation of the per-transaction trust and reliability values given in many feedbacks over a specified time period T_h .
- (2) *Total number of transactions performed*. A simple aggregation of feedbacks can misrepresent the true record of a *TTP-host's* transactional behavior and fail to capture any misbehavior by the *TTP-host*, should the total number of

- transactions that it has performed over a given period of time not taken into account. For example, a *TTP-host* that has performed dozens of transactions but cheated on 1 out of every 4 occasions will have a higher reputation value in comparison with a *TTP-host* that has only performed 10 transactions and has been faithful in all of these occasions. In other words, the total number of transactions that a *TTP-host* has performed over a specific past period is an important indicator of its reputation and should be taken into account for calculating and updating the reputation value of the *TTP-host*. Our model takes in the average feedback value, measured as the ratio of the sum of the feedbacks *TTP-host* has received over time period T_h to the total number of transactions in which the *TTP-host* has taken part over the same period.
- (3) *Transaction value.* The value of a transaction undertaken by a *TTP-host* is another important metric for its reputation evaluation. Undertaking a transaction with a value of £ 1000 certainly worth more credit than that with a value of £ 10. For example, the solution used by e-Bay cannot penalize a *TTP-host* that may develop a good reputation (with high accumulated feedback value) by being honest in performing small value transactions, but behaving maliciously with large value ones. Our reputation evaluation algorithm, therefore, is embedded with a risk factor that is dependant on the transaction value, and is used to weigh the feedback of the transaction for the *TTP-host*.
 - (4) *Total number of malicious incidents.* To further enhance the fairness in crediting and penalizing transactional behavior of a *TTP-host*, we have also introduced a counter for malicious incidents (outcomes). A *TTP-host* with this counter value reaching a certain threshold will have its reputation value reduced to the minimum. The *TTP-host* would have to perform a considerable volume of honest transactions in order to rebuild its reputation.
 - (5) *Source of the feedback.* The author in [20] stated that “When considering reputation information, we often account for the context and the source of the information”. Therefore, the feedback from parties (merchants) who have a better reputation should be weighed more in calculating reputation.

Measure 2. The distributed signcryption scheme proposed by Mu and Varadharajan [23] can be employed directly into our protocol to address security requirement (S2). However, this scheme imposes rather heavy computational loads on the agent owner side—approximately $(N + 4)$ exponentiation operations, where N is the number of *TTP-hosts* participating in the protocol execution. Moreover, if the agent owner is to play the role of a group manager in order to address security requirement (S1), the computational overhead will further increase to $(2N + 5)$ exponentiation operations. This high computational cost cannot be neglected if the protocol is to be applied in m-commerce applications or mobile computing arena, where the agent owner code is expected to execute in a mobile device with restricted computational power and storage capability. Therefore, we take an asymmetrical approach in distributing the computational load between the agent owner and the merchant host so that the agent owner will take less load while still satisfying requirements (S1) and (S2).

Measure 3. The concept of verifying partial signatures generated by *TTP-hosts* is applied. This concept differs from conventional signature verification in that it verifies the validity of a partial signature (generated using a proxy key share) through the use of a commitment generated by the signer, rather than the signer’s public key corresponding to the signature key used to generate a conventional signature. By using the commitment generated by the partial signature signer in the verification process, the verifier is able to evaluate the honesty of the signer and assign him/her a trust level that can then be used by our trust management scheme. Conventional signature verification through the use of signer’s public key does not provide this feature.

Measure 4. The threshold concept [30] is used to construct a complete proxy signature from multiple partial proxy signatures. Once at least F out of N (where $F \leq N$ is the threshold value) correct partial signatures have been received, signature recipient B can construct the complete proxy signature by combining these F partial signatures. In this way, the protocol can tolerate up to $(N-F)$ malicious or faulty *TTP-hosts*, which enhance the protocol’s robustness.

4. Distributed reputation management scheme

Two algorithms are needed to allow an agent owner to distribute a security-sensitive task among a set of N trusted hosts, *TTP-host_i*, where $i \in \{1, \dots, N\}$. The first algorithm, called TTP-hosts Subgroup Selection (TSS) algorithm, allows the agent owner to select a subgroup of Y most trustworthy *TTP-hosts* from N available ones based upon their trust and reliability values. The second algorithm, called Trust and Reliability Updating (TRU) algorithm, allows the agent owner to evaluate and assign trust and reliability values to each *TTP-host* that s/he has employed based upon the feedback received from his/her merchant host. In the remaining part of this section, we describe these two algorithms after giving the assumptions used for their design.

4.1. Assumptions

- The agent owner maintains a table TA (Trust Assessment) containing trust and reliability values associated with each of the *TTP-hosts* that the agent owner has dealt with in the past period T_h . As shown in Table 1, each row corresponds to one *TTP-host* containing six attributes: $\{TTP_i\text{-ID}, Trust_i, Reliability_i, Satisfaction_i,$

Table 1 An example of a single row in table TA

TTP_i -ID	$Trust_i$	$Reliability_i$	$Satisfaction_i$	$TotalTrans_i$	T-counter _{<i>i</i>}	R-counter _{<i>i</i>}
1. www.verisign.com.au 2. VeriSign Limited 3. IT Department 4. South Melbourne 5. Victoria 6. AU	2	3	2.5	10	0	1

Table 2 An example of a single row in table MR

Merchant _{<i>i</i>} ID	Reputation _{<i>i</i>}
1. www.dixons.co.uk	2
2. DSG Retail Limited	
3. Sales Department	
4. Hemel Hempstead	
5. Hertfordshire	
6. UK	

TotalTrans, T-counter, R-counter}. The TTP_i -ID is the unique identifier of host $TTP\text{-}host_i$, e.g. its distinguished name¹ [12]. $Trust_i$ and $Reliability_i$ are its trust and reliability values, respectively. The $Trust_i$ attribute indicates the level of $TTP\text{-}host_i$'s trustworthiness (or honesty) in performing its job. The $Reliability_i$ attribute indicates its reliability level, i.e. robustness in its service provision. We assume that the agent owner has no experience in dealing with the associated $TTP\text{-}host_i$ so the initial values of $Trust_i$ and $Reliability_i$ are set to zero (i.e. neutral). However, these initial values (zeros) are greater than those of a malicious $TTP\text{-}host$ (-5 in our scheme). In this way, a newly deployed $TTP\text{-}host$ will not be treated unfairly as a malicious one. $Satisfaction_i$ stores the average of both $Trust_i$ and $Reliability_i$, i.e. $Satisfaction_i = (Trust_i + Reliability_i)/2$. The higher the value of $Satisfaction_i$, the more confidence the agent owner has in the $TTP\text{-}host_i$. $TotalTrans_i$ refers to the total number of transactions in which the $TTP\text{-}host_i$ has taken part with the agent owner during the past period T_h . $T\text{-}counter_i$ is the total number of malicious incidences involved by $TTP\text{-}host_i$, and $R\text{-}counter_i$ is the total number unreliable incidences by $TTP\text{-}host_i$, during the period. There are N $TTP\text{-}hosts$ in the set, i.e. $i \in \{1, \dots, N\}$. We assume that table TA is sorted in a descending order according to the $Satisfaction$ value. This means that the $TTP\text{-}host$ with the highest $Satisfaction$ value shall be in the first row of the table. The agent owner may decide the upper-limit for the trust and reliability values according to his/her preferences.

- The agent owner also maintain a table MR (Merchant Reputation) containing reputation values associated with each of the merchants that the agent owner has dealt with in the past time period T_m . As shown in Table 2, each row corresponds to one merchant containing two attributes: $\{Merchant_i\text{-ID}, Reputation_i\}$. $Merchant_i\text{-ID}$ is the merchant's unique identifier, e.g. the distinguished name [12] of the merchant. $Reputaion_i$ specifies the level of reputation $Merchant_i$ has accumulated from its previous dealings with the agent owner in period T_m . The initial value assigned to $Reputation_i$ is zero (neutral), which means that the agent owner has no experience in dealing with $Merchant_i$. The value of $Reputation_i$ is updated by the agent owner as follows: $+1$ is added if the transaction outcome is positive, 0 if no response is received from the merchant, (this may be due to that the communication link is broken), or -1 if the transaction outcome is negative. Table MR in this case represents only the agent owner opinion on the merchants he has dealt with. Alternatively, table MR can be stored in a publicly accessible server so that the value $Reputation_i$ can be modified by any agent owner or customer, who has the experience with the merchant

¹ The distinguished name specified in [9] consists mainly of six fields: Common name (CN), Organisation Name (O), Organisation Unit (OU), Locality (L), State (S), and Country (C).

Table 3 An example of table TM

	Trust	Reliability
TTP ₁ -ID	No	Yes
TTP ₂ -ID	Unknown	No
...		
TTP _N -ID	Yes	Yes

and is authorized to do so. Hence Reputation_{*i*} represents the accumulated opinions about Merchant_{*i*} among the community. The choice of the type of information, a single agent owner’s opinion or a community’s opinion, decides the location of the MR table, which may locate either at the agent owner side or in a public server. We leave these decisions to users’ preferences. In our scheme, we adopt the first approach in which the agent owner stores the table MR locally. However, our scheme can easily accommodate and work with the second approach.

- The merchant, once agreed on a deal with the mobile agent, creates a table TM, containing the trust and reliability values for all the participating TTP-hosts. Each TTP-host has an entry in the table consists of three attributes: the first is the TTP-host’s ID, the second is its trust value and the third is its reliability value. The second and the third attributes will be set by the merchant host as follows: the second attribute will be assigned with one of the following values (*Yes*, *No*, *Unknown*) and the third attribute will be assigned with either ‘*Yes*’ or ‘*No*’, depending on the outcome of the transaction involved with the TTP-host. Table 3 gives an example of value settings for table TM. Table 4 summarizes example scenarios for the transaction outcome and the trust and reliability values associated to these scenarios.

The merchant fills table TM with the values correspond to the transaction outcome with each of the participating TTP-hosts. It then passes table TM to the agent owner, via the mobile agent, for him to update table TA accordingly.

- As mentioned above, we have specified validity periods T_h and T_m for the tables TA and MR, respectively. This can help the agent owner in maintaining the freshness of the relevant data and reduce memory and computational expenses.

Table 4 Possible values assigned to Trust and Reliability in various scenarios

Metric	Values	Scenarios
Trust _{<i>i</i>}	Yes	TTP-host _{<i>i</i>} sends a valid expected data to the merchant host.
	No	TTP-host _{<i>i</i>} sends an invalid expected data, or an unexpected data, or simply “a token that cannot pass a specified verification”, to the merchant host.
	Unknown	The merchant has not received a response from the TTP-host _{<i>i</i>} during the protocol run. Therefore, he cannot make judgment whether this TTP-host _{<i>i</i>} is trustworthy or not.
Reliability _{<i>i</i>}	Yes	The merchant host has received a response, either positive or negative, from TTP-host _{<i>i</i>} .
	No	The merchant host has not received any response from TTP-host _{<i>i</i>} during the protocol run even if repeated requests have been made. This may be due to that the TTP-host _{<i>i</i>} is out of service or the communication link between the merchant host and the TTP-host _{<i>i</i>} is broken down, etc.

Table 5 Examples of the values given to Thr1

Transaction value	Thr1
<£10	1
£10– £ 50	2
£50– £100	3
£100– £ 200	4
.....	..
£2000– £2100	20

4.2. TTP-hosts subset selection (TSS) algorithm

When an agent owner is to delegate a signature-signing task to his agent, the first thing s/he needs to do is to decide a subgroup of *TTP-hosts* that will take part in performing the task together with the agent. This subgroup is called the active subgroup (AS) hereafter. To select the AS, the agent owner specifies a reputation threshold value *Thr1* that is proportional to the total value of the transaction. Table 5 shows examples of the values that can be assigned to Thr1.

From Table 5, it can be seen that the higher the transaction value the higher the threshold *Thr1* should be. The owner then executes the TSS algorithm (to be mentioned next) that takes *Thr1* and table TA as its parameters. Starting from the top of the table, the algorithm selects one or more (a subset (>1)) *TTP-hosts* so that their aggregated Satisfaction value, computed using Eq. (1), is equal to or greater than *Thr1*.

$$Agg = \sum_{i=1}^j Satisfaction_i \quad (1)$$

where, *j* is the number of *TTP-hosts* in the AS list. The reason for comparing the aggregated average with *Thr1* is that if the Satisfaction value of the *TTP-host* in the first row, which is the highest in table TA as assumed earlier in Section 4.1, is equal to or greater than *Thr1*, then the level of confidence the agent owner has in this *TTP-host*, according to the transaction value, is enough to perform the transaction correctly. However, if the above condition is not satisfied, the agent owner has to find a subset (AS list) of *TTP-hosts* that the total of their Satisfaction values is equal to or greater than *Thr1* so that the agent owner will be confident that the AS members, collaboratively, can perform the transaction correctly.

The pseudo code for the TSS algorithm is as follows.

TTP-hosts Subset Selection (TSS) Algorithm

Input: table TA, Thr1

Output: AS member(s)

Method:

Initialize Agg to 0

For each row i in TA do

Compute Agg = Agg + Satisfaction_{*i*} /* the Satisfaction value for TTP-host_{*i*} is added to the aggregated average Agg */

If Agg equals to or greater than Thr1 then

```

Insert  $TTP_i$  -ID in AS list /*  $TTP-host_i$  is the sole
element in AS*/
Increment TotalTransi
Exit the loop
Else
Increment  $i$  and start the next loop iteration
    
```

By the end of the execution, the list AS will contain one or more *TTP-hosts*, which contributes to the aggregated Satisfaction value satisfying the threshold specified by the agent owner. As the members of the AS list are chosen as the most trusted and reliable among all *NTTP-hosts*, they are most likely to perform the transaction securely and reliably. In addition, according to the algorithm, the higher the transaction value, the higher the threshold *Thr1* will be set. As a result, the more *TTP-hosts* will be selected and take part in the transaction, and it would be more difficult for any single *TTP-host* to forge a proxy signature or manipulate the transactional process. The transaction outcome will be more likely to come to a satisfactory conclusion.

4.3. Trust and reliability updating (TRU) algorithm

This section describes the TRU algorithm an agent owner uses to update trust and reliability values for each *TTP-host* upon the completion of each transaction. The merchant, once agreed on a deal with the mobile agent, creates a table TM and fills it with the trust and reliability values for all the participating *TTP-hosts* depending on the transaction outcome, as described in Section 4.1. The merchant then passes table TM to the mobile agent.

Once the agent is back from the shopping trip, it submits the table TM to the agent owner. The agent owner refreshes the content of table TA according to the values received in table TM by executing the TRU algorithm that takes tables TM, TA and the merchant’s ID (Merchant-ID) as its input parameters. The algorithm performs a search to find a *TTP-host* in table TA that matches with the *TTP-host* in table TM. Once found, the algorithm updates the Trust and Reliability values associated with the *TTP-host* in TA according to the received values in TM and the reputation value $Reputation_k$ of the merchant $Merchant_k$ -ID (extracted from table MR) using the equations given in Table 6.

In Table 6, $NewTrust_i$ is the new value assigned to $Trust_i$ in table TA. $OldTrust_i$ refers to the aggregated $Trust_i$ value resulted from all previous transaction outcomes, i.e. it is the value maintained in table TA. The same interpretation is applied to $NewReliability_i$

Table 6 Equations for updating Trust and Reliability value in TA

	Values in TM	Updates in table TA
Trust _i	Yes	$NewTrust_i = OldTrust_i + ((\partial \times Reputation_k) / TotalTrans_i)$
	No	$NewTrust_i = OldTrust_i - ((\partial \times Reputation_k) / TotalTrans_i)$
	Unknown	$NewTrust_i = OldTrust_i$
Reliability _i	Yes	$NewReliability_i = OldReliability_i + ((\partial \times Reputation_k) / TotalTrans_i)$
	No	$NewReliability_i = OldReliability_i - ((\partial \times Reputation_k) / TotalTrans_i)$

and OldReliability_i . ∂ indicates the risk factor specified by the agent owner. It varies in proportional to the transaction value, which means that the amount of penalty or reward each $TTP\text{-host}_i$ receives is dependant on the value of the transaction. However, if the total number of untrustworthy or unreliable transactions, as indicated by T-counter $_i$ and R-counter $_i$, respectively, reaches a certain threshold $\text{Thr}2$, then the agent owner assigns the maximum penalty γ , e.g. $\gamma = -5$, to the values of Trust_i or Reliability_i . The pseudo code for the TRU algorithm is given as follows.

Trust and Reliability Updating (TRU) algorithm

Input: table TM, table TA, ∂ , Thr2, γ , Merchant-ID

Output: updated table TA

Method:

```

For each row k in MR
  Search for Merchant $_k$ -ID that matches Merchant-ID
  If Found then
    /* Fetch the reputation value associated with Merchant $_k$ -ID */
    Reputation = Reputation $_k$ 
  For each row i in TA
    Search for TTP $_i$ -ID that matches TTP $_i$ -ID in TM
    If Found then
      /* Update the Trust $_i$  value in table TA */
      If Trust $_i$ -TM is Yes then
        NewTrust $_i$ -TA = OldTrust $_i$ -TA + (( $\partial \times$  Reputation $_k$ ) / TotalTrans $_i$ )

      If Trust $_i$ -TM is No then
        Increment T-counter $_i$ 
        If T-counter $_i$  = Thr2 then
          Trust $_i$ -TA =  $\gamma$ 
        Else
          NewTrust $_i$ -TA = OldTrust $_i$ -TA - (( $\partial \times$  Reputation $_k$ ) / TotalTrans $_i$ )
      If Trust $_i$ -TM is Unknown
        NewTrust $_i$  = OldTrust $_i$ 
    /* Update the Reliability $_i$  value in table TA */
    If Reliability $_i$ -TM is Yes then
      NewReliability $_i$ -TA = OldReliability $_i$ -TA +
        (( $\partial \times$  Reputation $_k$ ) / TotalTrans $_i$ )
    If Trust $_i$ -TM is No then
      Increment R-counter $_i$ 
      If R-counter $_i$  = Thr2 then
        Reliability $_i$ -TA =  $\gamma$ 
    Else
      NewReliability $_i$ -TA = OldReliability $_i$ -TA -
        (( $\partial \times$  Reputation $_k$ ) / TotalTrans $_i$ )

```

An important feature of this algorithm is that the step value of award/penalty for Trust and Reliability values is not linear—it is inversely proportional to the total number of transactions performed by the $TTP\text{-host}$ concerned. In other words, the step

value gets smaller as the number of transactions performed by the *TTP-host* gets larger. In addition, the step value is also linked to the transaction value through risk factor ∂ . That is, the larger the transaction value, the more risks the agent owner needs to take, and the more reward/penalty the *TTP-host* will get shall the transaction succeed/fail. If a *TTP-host* repeatedly misbehaves or being unreliable, say, for *Thr*2 times, its Trust or Reliability values will drop to γ . Furthermore, the step value is weighed according to the reputation ($Reputation_k$) of the source of the feedback ($Merchant_k$). A feedback from a highly reputable merchant weighs more than that from a less reputable one. We believe that these features reflect users' expectation and acceptance in real life. The following example illustrates the case for supporting these features mentioned above. When a customer has performed tens or hundreds of correct and honest deals with a specific merchant, the customer will be less likely to stop dealing with this merchant if only a couple of deals with small values, i.e. small ∂ , have ended up in an unexpected way. However, ending up a transaction of a large value, i.e. large ∂ , with an undesirable outcome may completely put off the customer. This is why the effect of the outcome of a transaction on the overall Trust or Reliability value and subsequently the reputation of a *TTP-host* should be dependent on the three factors: the value of the transaction, the total number of the transactions that it has performed in the past period concerned, and the reputation of the merchant host. If a *TTP-host* is malicious or unreliable, it will have a bad track record (reflected by a low Trust or Reliability value) and it will have to take a large number of successful transactions for the Trust or Reliability value to be rebuilt up to an acceptable or previous level.

The two algorithms mentioned above jointly enable a non-linear reputation model that is more suited to real-life e-commerce scenarios. According to our best knowledge, there has not been any such non-linear reputation model proposed in the literature.

4.4. A working example

In this section, we illustrate how both TSS and TRU algorithms work and hence, to show the effectiveness of the Distributed reputation scheme by demonstrating all the steps through a working example. The example will show the state of the TA table before and after performing a single transaction by the agent owner.

We assume that the agent owner has dealt with four TTP-hosts in the past. Table 7 shows the information corresponding to each of the four TTP-hosts in table TA.

If the agent owner wants to delegate his agent to perform a transaction worth £ 150, then *Thr*1 will take the value 4. The TSS algorithm in this case will return the AS list, which contains only TTP_1 as this *TTP-host's* Satisfaction value is greater than *Thr*1. Consider that the agent owner wants to delegate his agent to perform a transaction

Table 7 Table TA prior to performing the transaction

TTP-ID	Trust	Reliability	Satisfaction	TotalTrans	T-counter	R-counter
TTP_1	8	12	10	5	0	1
TTP_2	6	10	8	10	4	0
TTP_3	5	5	5	7	0	0
TTP_4	3	3	3	2	1	1

Table 8 Table TM received from Merchant_i

TTP-ID	Trust	Reliability
TTP ₁	Yes	Yes
TTP ₂	No	Yes
TTP ₃	Unknown	No

worth £ 2000, hence $Thr1 = 20$. The outcome of executing TSS algorithm will be an AS list containing the first three *TTP-hosts* in table TA as their aggregated Satisfaction value (23) is greater than $Thr1$. The agent owner will set the values of the required parameters as follows: $Thr2 = 5$, $\delta = 4$, and $\gamma = -5$. When the agent returns from its journey, it submits a TM table, given by Merchant_i that the agent has performed the transaction with. Table 8 shows the possible contents of the received TM.

The agent owner will pick up Merchant_i's reputation value Reputation_i from table MR, which is assumed to be 6. Then the agent owner will update table TA by executing TRU algorithm. The updated TA is shown in Table 9.

5. Distributed signcryption with verifiable partial signatures (DiSigncryption) protocol

In this section, we apply the Distributed Reputation Management scheme presented in Section 4 to distributing the trusted TTP service employed in the Agent-based Threshold Proxy Signcryption (ATPS) protocol [2] to derive out novel DiSigncryption protocol. In other words, in the DiSigncryption protocol to be presented shortly, a set of multiple *TTP-hosts* are dynamically selected on per-transaction basis, and these *TTP-hosts* are used to play the role of the TTP jointly and collaboratively. In this way, the performance and security bottleneck problem caused by the use of a single *TTP-host* in the ATPS protocol is eliminated. In the following, we will first summarize the notation, and state the assumptions, used in the protocol design, and then present the protocol formally.

5.1. Notation

The notation to be used throughout the rest of this paper is summarized as follows.

- $H(x)$ is a one-way collision free hash function that takes a variable size input (x) and produces a fixed-size output (digest). An example of such a hash function is SHA-1 [24].

Table 9 Table TA after performing the transaction

	Trust	Reliability	Satisfaction	TotalTrans	T-counter	R-counter
TTP ₁	12.8	16.8	14.8	6	0	1
TTP ₂	-5	12.4	3.7	11	5	0
TTP ₃	5	1.58	3.29	8	0	0
TTP ₄	3	3	3	2	1	1

Table 10 Signcryption and unsigncryption operations

Signcryption by I the sender	Unsigncryption by j the recipient
1. i randomly chooses $t \in Z_q^*$, where p and q are large primes, q is a prime factor of $p - 1$ and g is an element of Z_q^* of order q . 2. i then calculates: $k = H((pk_j)^t \text{ mod } p)$ $y = g^x \text{ mod } p$ $c_1 = E_k(x)$ $r_1 = H(y, c_1)$ $s_i = t / (r_1 + sk_i) \text{ mod } q$ 3. i sends (c_1, r_1, s_i) to j	1. j computes from (c_1, r_1, s_i) : $y = (pk_i \times g^{r_1})^{s_i} \text{ mod } p$ $k = H(y^{sk_j} \text{ mod } p)$ $x = D_k(c_1)$ to obtain the plaintext message. 2. j accepts i 's signature if and only if $H(y, c_1) = r_1$

Table 11 DSS Signature Algorithm

DSS signature generation	DSS signature verification
1. i chooses a random number $k \in_R Z_q^*$ 2. $r = (g^k \text{ mod } p) \text{ mod } q$ 3. $s = (k^{-1}(H(x) + sk_i \times r)) \text{ mod } q$ The DSS signature is (r, s)	Check if $r = ((g^{H(x)s^{-1}})(pk_i^{rs^{-1}}) \text{ mod } p) \text{ mod } q$

- $E_k(x)$ and $D_k(x)$ express the encryption and decryption of a data item x using a symmetric key k and a symmetric cryptosystem such as Data Encryption Standard (DES) [25] or Advanced Encryption Standard (AES) [27].
- $Enc_{pk_i}(x)$ expresses the ciphertext of a data item x encrypted with the public key pk_i using ElGamal public-key cryptosystem [7].
- $SC_{i \rightarrow j}(x) (c_1, r_1, s_i)$ denotes the signcryption operation performed on a data item x using the private key of party i and the public key of party j . The triple (c_1, r_1, s_i) , produced by the signcryption operation, is the signcryption value of x . Any entity can verify i 's signature on x from the triple value but only j can unencrypt (decrypt c_1) the signcrypted x and reveal its plaintext. A brief description of the signcryption and unsigncryption operations [8] is given in Table 10.
 $[Sig_{sk_i}(x) = (r, s)$ denotes a digital signature on a data item x generated using a private key sk_i of party i and a signature scheme such as [26]. A brief description of the DSS algorithm is given in Table 11 below.
- sh_i denotes the proxy key share designated to $TTP\text{-}host_i$.
- (c_2, r_2, ps_i) denotes the partial signature generated by $TTP\text{-}host_i$ using a proxy key share sh_i , to be explained in Section 5.3.5.
- $Comm_i$ denotes a commitment generated by $TTP\text{-}host_i$ to authenticate its partial signature (c_2, r_2, ps_i) . Signature recipient B uses this commitment to check the validity of the received partial signature to ensure that it is indeed generated by using the correct key share sh_i .
- $A \xrightarrow{E} B: m$ denotes that party A sends party B a message m via an external channel such as a telecommunication network.

- $A \xrightarrow{I} B: m$ denotes that party A sends party B a message m internally via an internal message passing mechanism. This case applies when both A and B resides at the same host.

5.2. Assumptions

- Every party i ($i \in \{A, B, TTP - host_k\}$, and $k \in \{1, \dots, N\}$) has a pair of private and public ElGamal keys, expressed as $sk_i \in_R Z_q^*$ and $pk_i = g^{sk_i}$, where g is an element of Z_p^* of order q and p , and q are large primes such that q is a prime factor of $p-1$. The public key pk_i is certified in the form of a digital certificate $Cert(i)$ that is signed by a certification authority (CA) trusted by all parties.
- Parties A and B may not have mutual trust. That is, either of them may misbehave in order to gain some advantage over the other party. For example, party B may try to use A 's proxy key to sign more than one deal for which A will be held responsible. TTP -hosts are introduced in the protocol to assist proxy signature verification and to store transaction evidences for dispute resolution. It is assumed that TTP -hosts may misbehave or collude with each other, but F out of N TTP -hosts are trustworthy.
- The level of trust both A and B place on each of the TTP -hosts taking part in the protocol execution varies depending on the past experience A and B have in dealing with the TTP -hosts. A and B have each TTP -host $_i$'s public key certificate $Cert(TTP$ -host $_i)$. The TTP -hosts also have the public key certificates of the parties that have established a trust relationship with them, i.e. A and B in our protocol. These certificates will play a role in authentication and secure communications between these parties and the TTP -hosts.
- B is assumed to provide mechanisms to protect the mobile agents it hosts from being eavesdropped on their contents and execution flows by other agents hosted also by B . B can use existing solutions, e.g. tamper-resistant hardware [34] and time limited blackbox security [10], to provide such mechanisms.

5.3. Cryptographic primitives

The DiSigncryption protocol is designed by integrating the Distributed Reputation Management scheme presented in Section 4 with eight cryptographic primitives: Group Public Key Generation method, Proxy Key Generation method, Proxy Key Shares Generation method, Distributed Signcryption scheme, Partial Proxy Signature Generation method, Partial Proxy Signature Verification method, Complete Proxy Signature Generation method, and Proxy Signature Verification method. Following is a description of these primitives.

5.3.1. Group public key generation method [23]

This method is proposed by Mu and Varadharajan [23] to facilitate efficient distributed signcryption. In this method, a signer A signcrypts a message in such a way that only the designated receiver TTP -host $_i$, $i \in \{1, \dots, Y\}$, by using a group public key G and a secret key x_i known only to A and TTP -host $_i$, would be able to unsigncrypt the message. It is computational cost-effective as, for Y intended recipients, it need to signcrypt

the message only once instead of Y times as required by conventional signcryption method. In this method, A generates Y secrets $x_i \in \mathbb{Z}_p, i \in \{1, \dots, Y\}$, and send each x_i securely to the corresponding $TTP\text{-}host_i$. Only A and $TTP\text{-}host_i$ have the knowledge about the secret x_i . These secrets can be generated offline prior to protocol execution to reduce on-line computational cost. They can also be refreshed periodically for enhanced security. The method uses a polynomial function and a set of exponentials. The polynomial function of order Y is constructed as follows:

$$f(x) = \prod_{i=1}^Y (x - x_i) \equiv \sum_{i=0}^Y a_i x^i \pmod{q}, \quad (2)$$

where the set $\{a_i\}, i \in \{0, \dots, Y\}$ are coefficients with

$$\begin{aligned} a_0 &= \prod_{j=1}^Y (-x_j), \quad a_1 = \sum_{i=1}^Y \prod_{j \neq i} (-x_j), \dots, a_{Y-2} = \sum_{i \neq j, j}^Y (-x_i)(-x_j), a_{Y-1} \\ &= \sum_{j=1}^Y (-x_j), a_Y = 1. \end{aligned}$$

It is worth noting that $\sum_{i=0}^Y a_i x_j^i = 0$. This property is important for the design of the distributed signcryption protocol to be presented in Section 5.4. Having obtained the set $\{a_i\}$, A constructs the corresponding exponential values, i.e. the group public key $G = \{g^{a_0}, g^{a_1}, \dots, g^{a_Y}\} \equiv \{g_0, g_1, \dots, g_Y\}$. All elements are computed under modulo p . Note that $\prod_{i=0}^Y g_i^{x_j^i} = 1$.

5.3.2. Proxy key generation method

Once the group public key is generated, A uses the following proxy key generation method to generate a proxy key [1]. That is,

1. A generates two random numbers α and $\beta \in \mathbb{Z}_q$.
2. A computes $w = g^\beta$ (3)
3. A computes pr_A (i.e. the proxy key) $= (sk_A \times \alpha) + (\beta \times w) \pmod{q}$ (4)

A then keeps β as a secret and sends α and w securely to the $TTP\text{-}hosts$ participating in the protocol execution for the proxy signature verification purpose (will be explained in Section 5.3.8).

5.3.3. Proxy key shares generation method

As mentioned earlier, in the DiSigncryption protocol, the task of proxy signature generation is delegated to multiple entities, i.e. one agent and Y $TTP\text{-}hosts$. A therefore needs to use this proxy key share generation method to divide the proxy key pr_A into multiple shares. That is,

1. A generates $(Y + 1)$ shares $sh_i, i \in \{1, \dots, Y + 1\}$, of the key pr_A . For doing so, A first generates a $(F - 1)$ degree polynomial:

$$f_1(x) = \sum_{i=1}^{F-1} b_i x^i + pr_A \quad (5)$$

where the coefficients $b_i, i \in \{1, \dots, (F-1)\}$ are chosen randomly from Z_q and kept secret by A .

2. A then computes the shares sh_i by evaluating the polynomial $f_1(x)$ at $(Y + 1)$ different random points. These points will be made public after the share generation process. Therefore, instead of generating additional public values, A can use values that are already public and distinct, e.g. MA and TTP -hosts identities ID_s , for this purpose. Thus, A computes shares $sh_i = f_1(ID_i)$, where ID_i is the unique identity of MA and TTP -hosts, and their distinguished names [12] can be used as their identities.

5.3.4. Distributed signcryption scheme

Having got the proxy key shares ready, A sends each TTP -host $_i$, where $i \in \{1, \dots, Y\}$, a message M containing blinded proxy shares $sh_i, i \in \{1, \dots, Y\}$ (as explained next), a random number $rand \in Z_q$ (to be used for partial proxy signature generation), and other related items. To protect the confidentiality and authenticity of these shares, A uses the signcryption method detailed next to signcrypt M in such a way that TTP -host $_i$, using the group public G and its own secret key x_i , can only reveal its designated share sh_i .

1. A first blinds the shares $sh_i, i \in \{1, \dots, Y\}$ by computing:

$$u_i = (sh_i \times x_i) \quad (6)$$

2. A then signcrypts M by first choosing a random number $x \in Z_q$, which is kept secret by A , and then computes the following:

$$k = g^x \text{ mod } p \quad (7)$$

$$c_1 = E_k(M) \quad (8)$$

$$r_1 = H(c_1, k) \quad (9)$$

$$s_i = x / ((kr_1 + sk_A) \times u_i) \text{ mod } q, i \in \{1, \dots, Y\} \quad (10)$$

(c_1, r_1, s_i) represents the signcryption of M using the private key sk_A and the blinded secret key u_i . A then send the signcrypted message (c_1, r_1, s_i) together with the group public key $G\{g^{r_1 k} \times g^{a_0}, g^{a_1}, \dots, g^{a_Y}\} = \{g_0, g_1, \dots, g_Y\}$ and u_i to TTP -host $_i, i \in \{1, \dots, Y\}$.

3. Upon recipient of the signcryption (c_1, r_1, s_i) , G and u_i , TTP -host $_i$ authenticates the origin of (c_1, r_1, s_i) and reveals the signcrypted message M by performing the following operations.

- (i) Recover the key k by computing:

$$\begin{aligned} k &\leftarrow \left(pk_A g_0 \prod_{j=1}^Y g_j^{x_j^j} \right)^{s_i \times u_i} \\ &= \left(pk_A g^{kr_1} \prod_{j=0}^Y g_j^{x_j^j} \right)^{s_i \times u_i} = \left(pk_A g^{kr_1} \prod_{j=0}^Y g^{a_j x_j^j} \right)^{s_i \times u_i} \\ &= \left(pk_A g^{kr_1} g^{\sum_{j=0}^Y a_j x_j^j} \right)^{s_i \times u_i} \end{aligned} \quad (11)$$

as $\sum_{j=0}^Y a_j x_i^j = 0$ then

$$\begin{aligned}
 &= (pk_A g^{kr_1})^{s_i \times u_i} \\
 &= (g^{sk_A} g^{kr_1})^{\left(\frac{x}{(kr_1 + sk_A) \times u_i}\right) \times u_i} \\
 &= (g^{(sk_A + kr_1)})^{\left(\frac{x}{(kr_1 + sk_A) \times u_i}\right) \times u_i} \\
 &= g^x \\
 k &= g^x \pmod p \quad (12)
 \end{aligned}$$

- (ii) Confirm the correctness of A 's signature only if $H(c_1, k) = r_1$. By performing this check, $TTP\text{-}host_i$ will be convinced that A has indeed generated the signcryption as its public key pk_A is involved in the unsigncryption operation.
- (iii) Decrypt c_1 to reveal (M) by using the key k :

$$(M) = D_k(c_1) \quad (13)$$

- (iv) Unblind the share sh_i by computing:

$$sh_i = u_i / x_i \quad (14)$$

5.3.5. Partial proxy signature generation method

For Y entities to jointly generate a digital signature on a document Doc , these entities need to produce a partial signature each using the proxy key share mentioned in the section above. In addition, as mentioned in Section 3, we have devised an idea of using a commitment generated by the signature signer (instead of using its public key) for the partial signature verification. The following gives the details as how the partial signature (c_2, r_2, ps_i) and the commitment $Comm_i$ are generated by $TTP\text{-}host_i, i \in \{1, \dots, Y\}$. The commitment $Comm_i$, will be used by the signature receiver/combiner B to verify the partial signature to assure that (c_2, r_2, ps_i) has indeed been generated by using the correct share sh_i . Verifying the validity of the partial signature (c_2, r_2, ps_i) is necessary as, firstly, it enables B to exclude any invalid partial signature when constructing the complete proxy signature. Otherwise, the complete proxy signature will be invalid. In addition, the verification allows the detection of any malicious $TTP\text{-}host_i$ and penalizes it accordingly, as explained in Section 4.2. For a $TTP\text{-}host_i$ to generate a partial proxy signature and the corresponding commitment, it performs the following calculations.

$$k = H(pk_B^{rand} \pmod p) \quad (15)$$

$$y_1 = g^{rand} \pmod p \quad (16)$$

$$c_2 = E_k(Doc) \quad (17)$$

$$r_2 = H(y_1, c_2) \quad (18)$$

$$l_i = sh_i \prod_{j=1, j \neq i}^Y \frac{-ID_j}{ID_i - ID_j} \pmod q, \quad (19)$$

$$ps_i = rand / (r_2 + Yl_i) \bmod q \tag{20}$$

$$Comm_i = g^{x_i^{-1} \times \prod_{j=1, j \neq i}^Y \frac{-ID_j}{ID_i - ID_j}} \bmod p \tag{21}$$

Here, the partial signature on *Doc* is (c_2, r_2, ps_i) .

5.3.6. *Partial proxy signature verification method*

Upon receipt of a partial signature (c_2, r_2, ps_i) and the commitment $Comm_i$ from *TTP-host_i*, $i \in \{1, \dots, Y\}$, *B* verifies its validity to confirm that it has been generated by using the right key share sh_i . To do so, *B* first computes:

$$\begin{aligned} V &= Comm_i^{Y \times u_i} = \left(g^{x_i^{-1} \times \prod_{j=1, j \neq i}^Y \frac{-ID_j}{ID_i - ID_j}} \right)^{Y \times sh_i \times x_i} \\ &= g^{Y \times sh_i \times x_i \times x_i^{-1} \times \prod_{j=1, j \neq i}^Y \frac{-ID_j}{ID_i - ID_j}} = g^{Y \times sh_i \times \prod_{j=1, j \neq i}^Y \frac{-ID_j}{ID_i - ID_j}} \bmod p \end{aligned} \tag{22}$$

B then computes:

$$\begin{aligned} T &= (V \times g^{r_2})^{ps_i} = \left(g^{Y \times sh_i \times \prod_{j=1, j \neq i}^Y \frac{-ID_j}{ID_i - ID_j} + r_2} \right)^{ps_i} \\ &= g^{(Y \times sh_i \times \prod_{j=1, j \neq i}^Y \frac{-ID_j}{ID_i - ID_j} + r_2) \times \left(\frac{rand}{r_2 + Y \times sh_i \times \prod_{j=1, j \neq i}^Y \frac{-ID_j}{ID_i - ID_j}} \right)} \\ &= g^{rand} \bmod p \end{aligned} \tag{23}$$

Finally, *B* compares $H(T \bmod p, c_2)$ with r_2 , i.e. to confirm if (c_2, r_2, ps_i) is indeed generated with the right key share sh_i . If the outcome is positive, the honesty of *TT-host_i* is confirmed. Otherwise, *TTP-host_i* will be rated as untrustworthy. A retransmission request may be made by *B* in case the negative result is caused by channel errors.

5.3.7. *Complete proxy signature generation method*

Once *B* has received *F* out of *Y* valid partial signatures (the remaining $(Y-F)$ partial signatures can either be invalid if they fail to pass the verification detailed in section 5.3.5, or fail to arrive at all), it constructs the complete proxy signature S_{pr_A} by combining the *F* shares as follows:

$$S_{pr_A} = \left(F^{-1} \sum_{i=1}^F ps_i^{-1} \right)^{-1} \bmod p \tag{24}$$

The complete proxy signature is (c_2, r_2, S_{pr_A}) .

5.3.8. *Proxy signature verification method*

As *B* does not possess the items, i.e. α and w , required for the verification of the proxy signature (c_2, r_2, S_{pr_A}) (the items are intentionally distributed only to the *TTP-hosts* by *A* in order to achieve the property of non-repudiation of proxy signature receipt), *B* needs to

forward the newly constructed (c_2, r_2, S_{pr_A}) to the *TTP-hosts* for it to be verified. Once received this request, *TTP-host_i* performs the following operations to verify the proxy signature.

(i) Calculate $Q = pk_A^\alpha \times w^w$ (25)

(ii) Calculate $u = (Q \times g^{r_2})^{S_{pr_A}} \bmod p$ (26)

(iii) The proxy signature is valid if and only if $H(u, c_2) = r_2$

5.4. Disignryption protocol description

The DiSigncryption protocol has beautifully integrated the Distributed Reputation Management scheme presented in Section 4 and the cryptographic primitives presented in Section 5.3 to achieve distributed agent-based proxy signature delegation/generation. Here, in the following, the protocol is described as a seven-step procedure.

Step 1 – Execution initialization: During this stage, the agent owner specifies the shopping requirements and generates the parameters needed for the proxy signature delegation. In detail, *A* performs the following setup operations prior to the protocol execution.

1. *A* executes the TSS algorithm explained in Section 4.2, to select a subset of *TTP-hosts* from table TA, i.e. the *AS* list containing *Y* members (*Y* is an integer ranging from 1 to the maximum number of entries in TA), which would satisfy the risk threshold specified.
2. *A* uses the Group Public Key Generation method explained in Section 5.3.1 to generate a group public key *G* needed for the verification and decryption of a signcrypted message. That is, *A* first generates the set of coefficients $\{a_i\}, i \in \{1, \dots, Y\}$ and constructs the polynomial function (i.e. Eq. (1)). *A* then constructs the group public key $G = \{g_0, g_1, \dots, g_Y\}$ for the *Y* participating *TTP-hosts*.
3. *A* prepares a document M_A that specifies the purchase or signature generation requirements, e.g. description of goods to be purchased or the contract to be signed. M_A is signed with *A*'s private key sk_A using a conventional signature method, e.g. DSS [26], for authentication and integrity purpose.
4. *A* generates a proxy key pr_A from its private key sk_A by executing the Proxy Key Generation method (described in Section 5.3.2). *A* then constructs a message *M* containing the signed M_A along with items for proxy signature verification, i.e. the values to be used by the *TTP-hosts* for the verification of the proxy signature (α and w), the transaction identifier I_t , and the proxy key's validity period *V*, i.e. $M = (Sig_{sk_A}(M_A), \alpha, w, I_t, V)$.
5. *A* then generates $(Y + 1)$ shares $sh_i, i \in \{MA, 1, \dots, Y\}$ using the Proxy Key Share Generation method (described in Section 5.3.3). *A* also hashes the values of these shares, i.e. $SH = \{H(sh_1), H(sh_2), \dots, H(sh_Y)\}$ that will be used by the *TTP-hosts* to verify the integrity of the received shares. More details with regard to the use of hash values will be explained later in the protocol.
6. *A* signcrypts (M, SH) by executing the Distributed Signcryption scheme (Section 5.3.4). The resulting signcryption for (M, SH) is $(c_1, r_1, s_i), i \in \{1, \dots, Y\}$.
7. *A* loads *MA* with the following message:

T1. $A \xrightarrow{I} MA : ((ID_A, sh_{MA}, c_1, r_1, G, S, U, Sig_{sk_A}(sh_{MA}, c_1, r_1, G, S, U))$

Where $S = \{s_1, s_2, \dots, s_Y\}, U = \{u_1, u_2, \dots, u_Y\}$.

8. A then dispatches MA in to the network to search at various merchant hosts for a suitable offer. There are two ways for A to determine the IP addresses of the merchant hosts to be visited by MA : (1) through querying a service directory; or (2) through a manual search by the agent owner.

Remark. The above operations performed by A can be done offline, which reduces the computational load on A . This feature makes our protocol an ideal candidate for m-commerce applications where A is normally a resource-limited mobile device.

Step 2 – Offer searching and proxy key share distribution. MA now roams from merchant host to merchant host searching for an offer that satisfies the requirements specified in M_A . If such an offer is found at a merchant host, say M_B , then B will provide MA with an execution environment so that MA will run locally to execute the rest of the protocol. Residing at B , MA generates a random number $rand$ that will be used by MA and the Y TTP -hosts in the generation of the partial signatures (c_1, r_1, s_i) , $i \in \{1, \dots, Y + 1\}$. MA then sends the message $(c_1, r_1, s_i, rand, G, u_i, M_B)$ to each of the Y TTP -host via a secure channel, e.g. SSL [31], and sends U to B for it to perform partial signature verification via a message passing mechanism on the same host. That is,

$$\mathbf{T2.1.} \quad MA \xrightarrow{E} TTP - host_i : (c_1, r_1, s_i, rand, G, u_i, ID_B, M_B), i \in \{1, \dots, Y\}$$

$$\mathbf{T2.2.} \quad MA \xrightarrow{I} B : U$$

Step 3 – Partial proxy signature generation and delivery. Each TTP -host $_i$, once received message T2.1, verifies and decrypts it to recover the proxy key share and to use it to generate a partial proxy signature and its corresponding commitment. That is, the TTP -host $_i$ first performs the following verification:

Verification TTP-host-1

Check the correctness of A 's signature on the signcryption (c_1, r_1, s_i) received in T2.1. This is done by recovering the key k using Eqs. (11) and (12), and then checking if $H(c_1, k) = r_1$.

This verification is to ensure that the signcryption (c_1, r_1, s_i) , which contains items, α and w , required for proxy signature verification, is indeed generated by A , as A 's public key pk_A is used in the verification process. If the verification fails, TTP -host $_i$ sends an error message to MA asking it to resend the message T2.1. If the verification fails for the second time, TTP -host $_i$ will send an error message to both MA and B and terminate the protocol execution. Otherwise, if the verification is positive, TTP -host $_i$ proceeds by performing the following operations.

- Decrypts c_1 to reveal (M, SH) using Eq. (13).
- Unblinds the proxy key share sh_i using Eq. (14).

TTP -host $_i$ then performs the following verification:

Verification TTP-host-2

- (a) *Check the correctness of B 's signature on M_B using the DSS signature verification algorithm described in Table 7.*

- (b) Checks if the conditions specified in M_B matches with that in M_A contained in M
- (c) Check if the values, I_t , α , and w in M are fresh (i.e. they do not already exist in $TTP\text{-}host_i$'s database) and the time of the arrival of $T2.1$ is within the validity period V specified in M .
- (d) Verifies the integrity of the recovered proxy key share sh_i , i.e. check that the hash of the recovered share ($H(sh_i)$) exists in SH received in $T2.1$.

The purpose of Verification $TTP\text{-}host\text{-}2$ (d) is to detect any malicious attempt by B to make MA send invalid blinded share u_i instead of u_i by manipulating MA 's code. In details, $TTP\text{-}host_i$ uses A 's public key pk_A to decrypt c_1 (computes Eqs. 11–13) to reveal M and SH . This insures the $TTP\text{-}host_i$ that the shares' hashes in SH have been indeed generated by A . Therefore, the check (d) confirms that the recovered share sh_i from u_i is the original share generated by A and has not been manipulated by B .

If any of the steps in *Verification $TTP\text{-}host\text{-}2$* is negative, $TTP\text{-}host_i$ should send an error message to B and terminate the protocol execution. Otherwise, if the verifications is all positive, $TTP\text{-}host_i$ will compute a partial signature (c_2, r_2, ps_i) , $i \in \{1, \dots, Y\}$ on $Doc = (ID_A, ID_B, M_A, M_B, I_t)$ using Eqs. (15–20) of the Partial Proxy Signature Generation method (Section 5.3.5). $TTP\text{-}host_i$ also computes a commitment $Comm_i$ that will be used by B to verify the partial signature using Eq. (21).

The partial signature (c_2, r_2, ps_i) and the commitment $Comm_i$ are signed with $TTP\text{-}host_i$'s private key $sk_{TTP\text{-}host_i}$ and sent to B . Similarly, MA computes the partial signature (c_2, r_2, ps_{MA}) on Doc with its share sh_{MA} and sends it internally to B . That is,

$$T3.1. \quad TTP - host_i \xrightarrow{E} B: \begin{cases} ((c_2, r_2, ps_i, Comm_i), Sig_{sk_{TTP\text{-}host_i}}(c_2, r_2, ps_i, Comm_i)) \\ \text{Error message if either signature verification or} \\ \text{proxy key share verification failed} \end{cases}$$

T3.1 will be executed by all $TTP\text{-}host_i$ with $i \in \{1, \dots, Y\}$.

$$T3.2. \quad MA \xrightarrow{I} B : (c_2, r_2, ps_{MA})$$

Step 4 – Partial proxy signature verification and complete proxy signature construction: In this stage, merchant host B confirms the correctness of all the partial proxy signatures received and constructs the complete proxy signature provided at least F out of Y partial proxy signatures are correct. In other words, upon the receipt of message T3.1, B performs Verification B-1 defined as follows.

Verification B-1

- (a) Check the correctness of $TTP\text{-}host_i$'s signature on message T3.1 using the DSS signature verification algorithm described in Table 7.
- (b) Check the validity of the received partial proxy signatures using the Partial Proxy Signature Verification method explained in Section 5.3.6.

Verification B-1 (a) is to confirm the authenticity of both the partial signature (c_2, r_2, ps_i) and the commitment $Comm_i$. By verifying $TTP\text{-}host_i$'s signature on T3.1, B is convinced that (c_2, r_2, ps_i) and $Comm_i$ are indeed generated by $TTP\text{-}host_i$ and that they have not been altered during transfer. Verification B-1 (b) is to ensure that partial signatures (c_2, r_2, ps_i) , where $i \in \{1, \dots, Y\}$, have been generated with the right key share sh_i , where $i \in \{1, \dots, Y\}$. In other words, if this verification is successful for a particular partial signature,

then the generating host of this partial signature will be confirmed as trustworthy for this transaction and will get the credit accordingly.

Otherwise, if any of the above verifications fail, B will send an error message to the TTP -host concerned and request for a retransmission. If repeated negative verification results are obtained, then this TTP -host will be branded as “dishonest” and get penalty accordingly. That is, B will fill the table TM with the corresponding value for the trust and reliability attributes for each TTP -host $_i$. For those with positive verification results, the TTP -hosts will be classified as “honest” and the value of Trust will be set to ‘Yes’. Otherwise, the TTP -host(s) with negative verification results will get a ‘No’ for their/its Trust value. In addition, if no message is received from a particular TTP -host then it’s Trust value will be set to ‘Unknown’. The reliability value will be determined according to whether or not B has actually received (c_2, r_2, ps_i) at all. If B has received it, then the Reliability value will be set to ‘Yes,’ otherwise, it will be set to ‘No’.

In the case that B has received at least F out of Y valid partial signatures from Y TTP -hosts, (as mentioned earlier, the protocols can tolerate up to $(Y-F)$ incorrect or missing partial signatures), B will proceed to construct the complete proxy signature, (c_2, r_2, S_{pr_A}) , where S_{pr_A} is computed using Eq. (24) defined in the Complete Proxy Signature Generation method (Section 5.3.7). As B does not possess the items needed for the verification of the proxy signature, B has to forward the newly constructed proxy signature (c_2, r_2, S_{pr_A}) to the TTP -hosts, each of which will verify the signature and if positive, generate a Verification Token VT . It is worth noting that the verification items are intentionally distributed to the TTP -hosts only so that only the TTP -hosts can verify the validity of proxy signatures. In this way, a verified valid proxy signature also results in a witness, the signature verifier. Thus, B cannot later deny that a valid signature has been generated and handed over to it by MA unless B colludes with the TTP -host. This achieves the security property of non-repudiation of proxy signature receipt.

There are two approaches for B to request for signature verification by the TTP -hosts. In the first, B can send the proxy signature to every TTP -host $_i$, and each of them generates a VT_i (all VT_i s should be identical if all TTP -hosts executing the protocol correctly) and sends it back to B . B then chooses the VT that is identical to a majority (F out of Y) of the received VT s. In this way, a minority or invalid VT s can be excluded. In the second approach, B can choose the TTP -host that is most trustworthy and reliable, i.e. the one with ‘Yes’ to both its Trust and Reliability values, to perform the proxy signature verification task. A and B may decide or negotiate as which of the above options should be chosen.

$$T4. \quad B \xrightarrow{E} TTP - host_i : (I_1, c_2, r_2, S_{pr_A}, Sig_{sk_B}(I_1, c_2, r_2, S_{pr_A}))$$

Step 5 – Proxy signature verification and token generation. In this stage, TTP -host(s) perform proxy signature verification upon B ’s request, and generate a verification token (VT) accordingly, which proves that the deal has been signed and B has received MA ’s proxy signature.

Upon recipient of $T4$, TTP -host $_i$ performs the following verification.

Verification TTP-host-3

- (a) Check the correctness of B ’s signature on transaction $T4$ using the DSS signature verification method described in Table 7.

- (b) Check the correctness of the proxy signature (c_2, r_2, S_{pr_A}) using the Proxy Signature Verification method described in Section 5.3.8.
- (c) Check the freshness of the received proxy signature, i.e. ensure that transaction T_4 is not replied, by checking the database to see if the proxy signature of transaction I_t has not already been verified and does not already exist in $TTP\text{-}host_i$'s database) and that the arrival time of T_4 is within the validity period V specified by the agent owner.

If any of the above verification fails, $TTP\text{-}host_i$ will send an error message to B to request for retransmission. If repeated retransmissions still result in a negative verification outcome, $TTP\text{-}host_i$ will send an error message indicating that the proxy signature is invalid and terminates the protocol run. In this case, B will forward this error message to MA , which will be delivered in turn to A . If the verification is positive, which indicates both the validity of the signature (c_2, r_2, S_{pr_A}) and the authenticity of A 's delegation (by using A 's public key pk_A in the verification of the proxy signature), $TTP\text{-}host_i$ will generate and send to B , in Transaction T_5 , a signed and time-stamped verification token VT . This VT can be used to prevent B from false denial of signature receipt. Transaction T_4 together with VT will be used as a proof that B has indeed received the proxy signature thus supporting non-repudiation of the receipt of A 's signature by B .

$$T_5. \quad TTP - host_i \xrightarrow{E} B: \begin{cases} VT \\ \text{Error message if proxy signature verification failed} \end{cases}$$

Where,

$$VT = (c_2, r_2, S_{pr_A}, Sig_{g_{sk_B}}(c_2, r_2, S_{pr_A}), T_{TTP-host_i}, ID_A, ID_B,$$

$Sig_{g_{sk_{TTP-host_i}}}(c_2, r_2, S_{pr_A}, Sig_{g_{sk_B}}(c_2, r_2, S_{pr_A}), T_{TTP-host_i}, ID_A, ID_B))$ and $T_{TTP-host_i}$ is a timestamp generated by $TTP\text{-}host_i$ to indicate the time when the VT is generated.

Step 6 – Signing and returning the token to MA. Merchant B , once obtained the verification token VT , confirms $TTP\text{-}host_i$'s signature on VT . If the verification is positive, B is convinced of the authenticity of the token. B then submits the VT and table TM to MA that will return back to its owner A . If B receives an error message in T_5 , it forwards this error message to MA .

$$T_6. \quad B \xrightarrow{I} MA: \begin{cases} (VT, TM, Enc_{pk_A}(rand), Sig_B(VT, TM, Enc_{pk_A}(rand))) \\ \text{Error message} \end{cases}$$

Step 7 – Protocol execution completion and final verification. In this stage, MA returns home and passes the signed deal to its owner A . If A receives the signed VT in T_6 rather than the error message, A performs the following verification to confirm the signed deal:

Verification A

- (a) Check the correctness of B 's signature on transaction T_6 using the DSS signature verification method described in Table 7.
- (b) Check the correctness of $TTP\text{-}host_i$'s signature on VT using the DSS signature verification method described in Table 7.

(c) Check the correctness of the proxy signature S_{pr_A} in VT using the Proxy Signature Verification method described in Section 5.3.8.

The purpose of Verification A (a) is to authenticate the origin of $T6$ and to ensure its integrity. The purpose of Verification A (b) is to ensure the validity of the deal, and especially, B 's signature on Doc . The purpose of Verification A (c) is to ensure that B has sent a correct VT .

Verification A being positive means that A has conducted a valid deal with B , and both A and B have received a valid document Doc signed by both parties. A , in this case, retrieves the plaintext Doc from the signcryption (c_2, r_2, S_{pr_A}) in VT by performing the following tasks:

1. A decrypts the ciphertext $Enc_{pk_A}(rand)$ with the private key sk_A to reveal $rand$.
2. A then uses $rand$ together with B 's public key pk_B to generate k using Eq. (15).
3. A finally uses k to decrypt the ciphertext c_2 and reveal the plaintext Doc .

If the outcome of Verification A is positive, A stores the VT as it has both his and B 's signatures on Doc , approved by a TTP -host. A also updates the table TA according to the data in table TM by executing TRU algorithm in Section 4.3.

If Verification A is negative, which means that A has failed to obtain a correct VT , A initiates a recovery protocol with a TTP -host to recover the VT . A deadline T_d agreed by both A and B can be introduced into the above DiSigncryption protocol to specify when the signed VT should arrive. In this case, A can also initiate the recovery protocol when the deadline is passed before receiving any signed VT , i.e. before the agent has returned from its journey. A detailed description of the recovery protocol can be found in [2]. Figure 1 describes the basic DiSigncryption protocol (excluding the recovery protocol). A summary of the methods/algorithms performed by each party in the protocol with their inputs and outputs is given in Table 12.

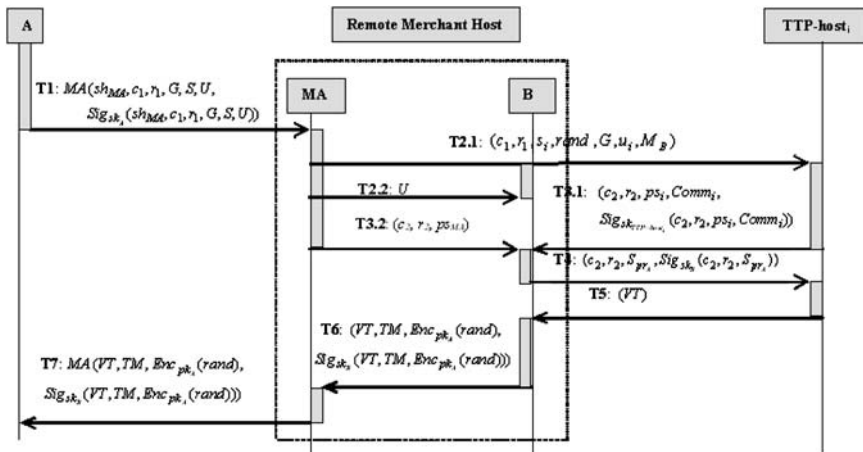


Fig. 1 Distributed signcryption with verifiable partial signatures (DiSigncryption) protocol

Table 12 Summary of the methods/algorithms performed by each party with their inputs and outputs

Party	Method/Algorithm	Input	Output
A	TSS Algorithm	TA, Thr_1	AS
	Group Public key Generation Method	–	$x_i, G, i \in \{1, \dots, Y\}$
	Proxy Key Generation Method	sk_A, α, β	pr_A
	Proxy Key Share Generation Method	$pr_A, ID_i, i \in \{MA, 1, \dots, Y\}$	$SH = \{sh_1, \dots, sh_Y\}$
	Distributed Signcryption Scheme – 1 & 2	SH, x_i, M, sk_A	c_1, r_1, s_i, U
	TRU Algorithm	$TM, TA, \delta, \gamma, Thr_2$	TA
MA	Partial Signature Generation Method	$pk_B, rand, Doc, sh_{MA}, ID_j, i \in \{MA, 1, \dots, Y\}$	(c_2, r_2, ps_{MA})
B	Partial Proxy Signature Verification Method	$Comm_i, u_i, c_2, r_2, ps_i$	True/False
	Complete Proxy Signature Generation Method	$ps_i, i \in \{1, \dots, Y\}$	c_2, r_2, S_{pr_A}
TTP -host _{i}	Distributed Signcryption Scheme – 3	pk_A, r_1, s_i, u_i	True/False
	Distributed Signcryption Scheme – 4	$pk_A, c_1, r_1, x_i, s_i, u_i$	M, sh_i
	Partial Signature Generation Method	$pk_B, rand, Doc, sh_i, ID_j, i \in \{MA, 1, \dots, Y\}$	$(c_2, r_2, ps_i), Comm_i$
	Proxy Signature Verification Method	$c_2, r_2, S_{pr_A}, pk_A, \alpha, w, Doc$	True/False

6. Analysis of the Disigncryption protocol

6.1. Comparison with related work

To highlight the merits of our DiSigncryption protocol, the efficiency, reliability, robustness, and accountability of our protocol is compared with that of both single TTP -host based and distributed (or multiple) TTP -hosts based protocols. The ATPS protocol proposed in [2] is chosen as an exemplary single TTP -host based approach and the recently proposed protocols in [11], hereafter referred to as Hsu’s protocol, and in [33], referred to as Tzeng’s protocol, are chosen as samples of the distributed TTP -host based approach. The reason for choosing these protocols is that they are all designed to perform the same task as ours. That is, an original signer delegates its signing power to proxy signer(s). The only difference between the two groups of protocols is that the ATPS protocol employs a single TTP -host to assist an MA to generate a proxy signature and the other two protocols, i.e. Hsu’s and Tzeng’s protocols, employ multiple TTP -hosts as proxy signers. In both Hsu’s and Tzeng’s protocols, the proxy key is generated by the original signer and shared among a group of N proxy signers. Each proxy signer can then generate a partial proxy signature on the desired message. Any T or more out of the N proxy signers can cooperatively reconstruct and verify the proxy signature on the message, but $(T - 1)$ or fewer proxy signers cannot.

Table 13 shows the comparison between the DiSigncryption protocol and the three related works mentioned above in terms of communication overheads, measured in terms of the number of messages exchanged among the protocol entities and their sizes. The following assumptions have been used in calculating a message size:

- AES algorithm [27] is used for symmetric encryption, therefore, the size of a ciphertext is in multiples of 128 bits.
- The key used for AES is 192 bits long.

Table 13 Communication overhead

	Total number of messages exchanged between the protocol parties	Total number of messages	Total size of messages (in bytes)
ATPS protocol	9	9	6200
Hsu's protocol	$2N^2 + 2N + 3T^2 - 3T$	280	75440
Tzeng's protocol	$N^2 + 2N + 2T^2 - T$	165	18800
DiSigncryption protocol	$5Y + 3$	28	32269

- SHA-1 algorithm [24] is used for one-way hashing, therefore, the output of the hashing process $|H(\cdot)|$ is 160 bits.
- The prime $|p| = 1024$ bits, $|q| = 160$ bits
- The identifiers used in the protocol, e.g. ID_A , are 32 bits in size.
- The sizes of the agent owner's requirements M_A and the remote host's offer M_B are 512 bytes each. Therefore the size of $Doc = (ID_A, ID_B, M_A, M_B, I_t)$ is 1024 bytes.

Table 14 shows the computational overhead measured in terms of the total number of multiplication and exponentiation operations performed at each protocol step. The quantitative results in both tables are materialized by evaluating them with the total number of participating *TTP-hosts* (N) = 10, the minimum threshold of active *TTP-hosts* (T) = 5, and the number of *TTP-hosts* in the *AS* list (Y) = 5.

From Table 13, it can be seen that the communication overhead of the DiSigncryption protocol is approximately 5.2 times more than that of the ATPS protocol. From Table 14, it can be seen that the exponentiation operations (considered as the most resource-consuming operations) of the DiSigncryption protocol is approximately 5.4 times more than that of the ATPS protocol. The extra cost comes from the fact that the role of the TTP is distributed among Y *TTP-hosts* rather than played by a single TTP and, hence, the protocol requires communications among the Y *TTP-hosts*, which is not needed when only one *TTP-host* is used. However, the downside of using the single *TTP-host* based approach is that it is less reliable and more prone to becoming a performance bottleneck. If this *TTP-host* is busy or goes down, the protocol execution would be affected or come to a halt. In addition, as all the sensitive information related to a transaction is stored at the *TTP-host* for the provision of non-repudiation of receipt service and dispute resolution, any security compromise at the *TTP-host* will have serious security and privacy consequences to all the parties that uses the TTP service. The multiple *TTP-hosts* approach, on the other hand, used in the DiSigncryption protocol alleviates these weaknesses by allowing multiple *TTP-hosts* to jointly play the role of the TTP service. Up to $(Y-F)$ *TTP-hosts'* malfunction or untrustworthiness can be tolerated. In other words, our protocol is designed to work properly as long as a subset of at least F out of Y *TTP-hosts* works properly. In general, the use of a distributed approach brings two main advantages in comparison with the single *TTP-host* based approach. Firstly, it provides an increased level of security, as more than one *TTP-host* would have to be compromised in order to obtain the secret information, e.g. a proxy key. Secondly, it provides an increased level of reliability, as a protocol execution can still continue as long as a certain number of *TTP-hosts* perform properly (or only a limited number of them are malfunctioning or out of service). These advantages come at a

Table 14 Computation overhead

	Proxy key generation and sharing		Partial proxy signature generation		Proxy signature generation and verification		Total	
	Mult	Exp	Mult	Exp	Mult	Exp	Mult	Exp
ATPS protocol	7	9	7	14	9	14	23	37
Hsu's protocol	$3N + 3NT + 1$	$2N^2T - 2N^2 + 3N + 5NT + 2$	$2T^3 + 2T^2 + 3T + 3$ NT	$7T^2 + 10T$	$5T + NT$	4T	721	1327
Tzeng's protocol	$N^2T + NT + 2N + 1$	$N^2T + 3NT + N + T + 1$	$T^3 + T$	$T^3 - T^2 + T$	$5T^2 + 7T + NT$	$7T^2 + 7T$	911	981
DiSigmryption protocol	$Y^2 + 5Y + 8$	$N + Y^2 + 2Y + 7$	$2Y^2 + 11Y + 2$	$2Y^2 + 7Y + 1$	$9Y + 17$	$10Y + 12$	227	200

price. Extra communication and computation overheads are needed as the result of the communication needs among the multiple *TTP-hosts*.

In comparison with Hsu's and Tzeng's protocols, our DiSigncryption protocol achieves 90% and 77% reduction in the number of messages exchanged, respectively. This is a very impressive saving and is an advantageous feature when applied in mobile network environment, which is characterized by low and/or expensive bandwidth and higher error rate. In addition, the total size of our protocol messages is approximately 57% less than that of Hsu's scheme. However, the total size of our protocol messages is 41% higher than that of Tzeng's protocol due to the fact that our protocol provides four extra security services, non-repudiation of signature receipt, fairness for both the agent owner and the merchant, confidentiality of the signed message, and the accountability of *TTP-hosts* service, which neither Hsu's nor Tzeng's protocol provides. These features/services are necessary for e-/m-commerce applications. Regarding the computational costs, the DiSigncryption protocol enjoys a saving of approximately 84% and 79% in the number of performed exponentiation operations, and 68% and 75% in the number of performed multiplication operations, comparing with Hsu's and Tzeng's protocols, respectively. This savings are mainly due to the following reasons:

- Our protocol makes use of a smaller but most trustworthy sub-set (Y) of the N *TTP-hosts* rather than using the entire set of the N *TTP-hosts* indiscriminately as the other two protocols.
- The N *TTP-hosts* in both Hsu's and Tzeng's protocols mutually generate the proxy key shares, whereas in the DiSigncryption protocol, the proxy key shares is generated solely by the original signer.
- T *TTP-hosts* in both Hsu's and Tzeng's protocols mutually reconstruct the proxy signature. In our protocol, the proxy signature reconstruction is performed by the signature combiner, i.e. the merchant B .

Regarding the employed cryptographic primitives, our protocol is based on the "Signcryption" cryptographic primitive, which provides both message authenticity and integrity (signature), and confidentiality (encryption). This is done with a computational cost less than performing the signature and the encryption separately [8]. However, Hsu's and Tzeng's protocols are based on Elgamal-like digital signatures, which provide only message authenticity and integrity.

One may argue that increasing the number of the *TTP-hosts* in the protocol enhances the protocol's robustness and reliability, and therefore, reducing the number of the *TTP-hosts* might weaken these properties. This claim can be thwarted in our protocol by the fact that the Y participating *TTP-hosts*, i.e. the members of the *AS* list, are chosen based upon their trust level, i.e. their reputation. The Y *TTP-hosts* are the most trustworthy and reliable among the N *TTP-hosts*, and they are more likely to perform properly and reliably. In other words, our protocol is embedded with a real-time trust and reliability evaluation scheme by which the original signer can choose the most trustworthy and reliable subset Y among N *TTP-hosts*. In Kim's and Sun's protocols, all the *TTP-hosts* are used indiscriminately for each protocol execution. In addition, the real-time trust and reliability evaluation mechanism used in our protocol is able to detect a well-behaved/ misbehaved *TTP-host* and provides a measure to credit/penalize it accordingly. This feature, to the best of the authors' knowledge, has not been seen in the literature.

6.2. Security analysis

In this section, we analyze the security properties of the DiSigncryption protocol demonstrating that it satisfies all the security requirements stated in Section 3.

- Proxy key confidentiality: It is difficult to compromise the proxy key pr_A due to the following reasons: (1) the proxy key is distributed in $(Y + 1)$ shares, and (2) each share is blinded with a secret x_i that is known only to A and the $TTP\text{-}host_i$. In order to compromise the proxy key, one has to intercept and brute force attack at least F blinded shares u_i , of which the security relies on the difficulty of factoring large primes unless F or more $TTP\text{-}hosts$ collude together.
- Proxy key shares confidentiality: Each share sh_i is blinded by the secret x_i , and both A and $TTP\text{-}host_i$ are the only parties that have knowledge of x_i , so only A and $TTP\text{-}host_i$ have knowledge of this share. The security of u_i depends on the difficulty of factoring large primes, i.e. factoring u_i to get sh_i and x_i .
- Proxy signature unforgeability: Since the proxy key pr_A is derived from A 's private key sk_A , it would be difficult for another party to forge the proxy key without knowledge of A 's private key. In addition, as multiple $TTP\text{-}hosts$ are involved in the proxy signature generation process, each $TTP\text{-}host$ has only a share of the proxy key, and this share can only be used to generate a partial signature, it would be difficult for a single $TTP\text{-}host$ to forge a valid proxy signature on Doc without colluding with others.
- Partial proxy signature verifiability: B is able to verify the validity of each partial proxy signature (c_2, r_2, ps_i) using the commitment $Comm_i$ generated by $TTP\text{-}host_i$ and the corresponding u_i received from A through the mobile agent MA . It is worth noting that B is able to verify (c_2, r_2, ps_i) without accessing plaintext sh_i . This feature supports the confidentiality of the proxy key shares and hence protects the proxy key from being disclosed to any non-holding parties including B .
- Non-repudiation of signature origin: The verification *Verification TTP-host-1* performed by the $TTP\text{-}hosts$ ensures that the proxy signature S_{pr_A} on Doc is generated by using a proxy key that is generated from A 's private key, and that the proxy signature verification requires the use of A 's public key. Therefore, A cannot deny the fact that he has generated the proxy key.
- Non-repudiation of signature receipt: This requirement is achieved through the use of a verification token VT signed by the $TTP\text{-}hosts$ and sent to A through MA in T6. As B cannot verify the proxy signature, B has to send a signature verification request to the $TTP\text{-}host_i$, which proves that B has actually received A 's proxy signature on Doc if the verification is positive and the token VT , signed by the $TTP\text{-}host_i$ is produced. Therefore, B cannot deny later that it has received A 's proxy signature on Doc .
- Fairness: Our protocol achieves the fairness requirement, i.e. by the end of a protocol execution, either both A and B have received the signed document Doc or none of them has anything to prove that the deal is done. B , after receiving VT from the $TTP\text{-}hosts$ in T5, may attempt to cheat A by sending an incorrect VT (for a deal between B and another party Z , for example) or a bogus message to MA . In other words, B may attempt to get A 's signature (verified proxy signature) on Doc but refuse to hand out its own one. A can discover this attempt when performing *Verification A* and consequently initiate the recovery process with a $TTP\text{-}host$ [2] to retrieve VT .

- Restricting the misuse of a proxy key: There are three scenarios where B might misuse a proxy key carried by an agent MA :
 - (1) B tampers with MA 's code, e.g. making the agent skip the verification performed in Step 2 so as to trick the MA to accept an offer M_B' that does not comply with M_A .
 - (2) B attempts to make MA signs more than one valid offer with the same proxy key so as to overcharge A for multiple deals.
 - (3) B eavesdrops on MA 's content, and copy and save the proxy key share sh_{MA} and the other shares $sh_i, i \in \{1, \dots, Y\}$ (which are signcryptured in (c_1, r_1, s_i) and blinded in U) for forging deals in the future. B may do this, for example, when B has no suitable offer for MA at the moment. When performing this attack, B sends T2.1 to $TTP-host_i$ by masquerading as MA , and thereby making A responsible for these signatures.

The first scenario is prevented in our protocol by letting the $TTP-host_i$ to perform *Verification TTP-host-2 (c)*. Should B launch the attacks described in the first scenario, it will only get a partial signature (c_2, r_2, ps_{MA}) on M_B' , which is useless on its own. The $TTP-host_i$ will discover this fraudulent attempt when executing *Verification TTP-host-2 (b)* and stop the protocol run without providing B with other partial signature(s) (c_2, r_2, ps_i) . To launch a successful attack as described in the second scenario, B has to obtain both MA 's and at least F valid $TTP-hosts'$ partial signatures $(ps_{MA}$ and $ps_i, i \in \{1, \dots, F\})$ on another valid document Doc'' . It is difficult for B to get hold of F proxy key shares $sh_i, i \in \{1, \dots, Y\}$ as each of them is blinded with a secret x_i that is only known to A and $TTP-host_i$. Without these key shares, it is difficult for B to forge the partial signatures. In addition, the values I, α , and w in the newly received T2.1 will undergo a freshness check, as defined in *Verification TTP-host-2 (c)*. If these values are replayed, $TTP-host_i$ will discover it and terminate the protocol execution. The third type of attacks can be easily detected by the $TTP-host_i$ when performing *Verification TTP-host-2 (c)* since the time of the arrival of T2.1 sent by B is likely to exceed the validity period V specified in M , which will cause the $TTP-host_i$ to terminate the protocol execution.

- Confidentiality of the document to be signed: The confidentiality of document Doc while it is being transferred between protocol's parties (A, MA, B , and $TTP-hosts$) is achieved by using the following measures.
 - (1) The communication channels between B and $TTP-hosts$ are secured using SSL that protects the confidentiality of the messages transmitted through them.
 - (2) The items sent through the channels between A and B , i.e. T1 and T6, are secured using signcryption. In T1, M_A , which is part of Doc , is sent signcryptured with the private key sk_A and the secret x_i in the form (c_1, r_1, s_i) . Only $TTP-host_i$, who knows x_i , can unsigncrypt (c_1, r_1, s_i) and reveal M_A . Doc in T6 is signcryptured with the proxy key pr_A , and to reveal Doc , one has to get access to either pr_A or $rand$. The confidentiality of pr_A has already been proven previously; and $rand$ is encrypted with A 's public key, i.e. only A can decrypt it with its private key sk_A . Therefore, Doc is protected over this channel.
 - (3) The items sent through the channels between MA and B , i.e. T2.2, T3.2, and T6, are either contain no useful information regarding the contents of Doc or secured using signcryption. In details, T2.2 contains no sensitive information regarding Doc as it contains only the blinded shares U . Messages T3.2 and T6 contain a signcryption of Doc with the public key pk_B and the proxy key share sh_{MA} for T3.2 and the proxy

key pr_A for T6. This insider, in addition to knowing T3.2 and T6, needs to know sk_B and $rand$ in order to reveal Doc from these messages. sk_B is protected by B and $rand$ is kept by MA and is not sent in plaintext to any of the protocol's parties, including B . The security of MA 's execution environment in terms of protecting the confidentiality of MA 's contents, e.g. prevention of $rand$ from being disclosed to a malicious insider, has been discussed in Section 5.2. With these measures, the confidentiality of Doc is protected over this channel.

- *TTP-host* accountability: This security requirement is addressed through the use of Verification B-1 and our proposed award/penalty mechanism. The outcome of Verification B-1 performed by B in the verification of the partial proxy signature (c_2, r_2, ps_i) received from $TTP-host_i$ will indicate if $TTP-host_i$ has followed the protocol execution correctly and credit/penalize it accordingly.

7. Conclusion

This paper has addressed the distributed reputation management issue by critically analyzing related works and highlighting their shortcomings. We then presented a novel Distributed Reputation Management scheme, which enables a party A , i.e. a customer, to distribute a security sensitive task among several *TTP-hosts*. This is achieved by first choosing a subset of *TTP-hosts* with the highest trust and reliability levels. The scheme then credits/penalizes each *TTP-host* according to a feedback received by A from party B , e.g. a merchant. The paper then presented a novel DiSigncryption protocol, which integrates into it the Distributed Reputation Management scheme, a modified version of the Distributed Signcryption proposed in [23] and an extended version of the ATPS protocol proposed in [2]. The new protocol has the following features. Firstly, it enables the agent owner to delegate signing power to its mostly trusted subset of *TTP-hosts* depending on the risk level valued upon transaction values. Secondly, it enables the signature combiner, i.e. the merchant, to verify each partial signature received from a *TTP-host* without access to the plaintext proxy key share, and verification outcome is used to rate the *TTP-host*'s honesty and credit/penalize it accordingly. Thirdly, it has an embedded algorithm to allow the agent owner to update the trust and reliability values for each *TTP-host*. The protocol analysis shows that, in addition to fulfilling the security requirements specified in Section 3, it is more flexible and robust 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. It is worth noting that the Distributed Reputation Management scheme can be used on its own in any system needed to manage the trust distributed among several hosts performing security sensitive tasks.

The future work will be the formal verification of the security properties of the proposed protocol.

References

- [1] Bamasak, O., & Zhang, N. (2004). A secure method for signature delegation to mobile agents. In *Proceedings of the 19th ACM Symposium on Applied Computing*, (pp. 813–818). Nicosia, Cyprus: ACM Press.
- [2] Bamasak, O., & Zhang, N. (2004). A secure proxy signature protocol for agent-based M-commerce applications. In *Proceedings of the 9th IEEE Symposium on Computer and Communications*, (pp. 399–406). Egypt, Alexandria: IEEE Computer Society.

- [3] Cachin, C. (2001). Distributing trust on the internet. In *Proceedings of International Conference on Dependable Systems and Networks (DSN2001)*, Gteborg. (pp. 183–192). Sweden, IEEE Computer Society.
- [4] Desmedt, Y. (1988). Society and group oriented cryptograph: a new concept. In *Advances in Cryptology, proceedings of Crypto' 87*, Lecture Notes in Computer Science, vol. 293 (pp. 120–127). Springer-Verlag.
- [5] Desmedt, Y., & Frankel, Y. (1992). Shared generation of authenticators and signatures. In *Advances in Cryptology, Proceedings of Crypto' 91*, Lecture Notes in Computer Science, Vol. 576 (pp. 457–469). Springer-Verlag.
- [6] eBay. <http://www.ebay.com>.
- [7] ElGamal, T. (1985). A public key cryptosystem and signature scheme based on discrete logarithms. *IEEE Transaction on Information Theory*, 31, 469–472.
- [8] Gamage, C., Leiwo, J., & Zheng, Y. (1999). Encrypted message authentication by firewalls. In *Proceedings of the Second International Workshop on Practice and Theory in Public Key Cryptography (PKC'99)*, Lecture Notes in Computer Science, Vol. 1560 (pp. 69–81). Springer-Verlag.
- [9] Harn, L. (1994). Group oriented (t, n) digital signature scheme and digital multisignature. *IEE Proceedings—Computers and Digital Techniques*, 141(5), 307–313.
- [10] Hohl, F. (1998). Time limited blackbox security: Protecting mobile agents from malicious hosts. In *Mobile Agents and Security*, Lecture Notes in Computer Science, Vol. 1419 (pp. 92–113). Springer-Verlag.
- [11] Hsu, C., Wu, T., & Wu., T. (2003). Improvement of threshold proxy signature scheme. In *Applied Mathematics and Computation*, Vol. 136, (pp. 315–321) Elsevier Science.
- [12] ITU-T. (1997). ITU-T recommendation X.500. ISO/IEC 9594-1, Information technology—Open Systems Interconnection—The Directory: Overview of concepts, models and services.
- [13] Jøsang, A., & Ismail, R. (2002). The beta reputation system. In *Proceedings of the 15th Bled Electronic Commerce Conference*, Bled, Slovenia.
- [14] Kim, S., Park, S., & Won, D. (1997). Proxy signatures, revisited. In *Proceedings of the International Conference on Information and Communications Security (ICICS' 97)*, Lecture Notes in Computer Science, Vol. 1334 (pp. 223–232). Springer-Verlag.
- [15] Kim, H., Baek, J., Lee, B., & Kim, K. (2001). Secret computation with secrets for mobile agent using one-time proxy signature. In *Proceedings of the 2001 Symposium on Cryptography and Information Security (SCIS2001)*, (pp. 845–850) Osio, Japan.
- [16] Kotzaniolaou, P., Burmester, M., & Chrissikopoulos, V. (2000). Secure transactions with mobile agents in hostile environments. In *Proceedings of the Fifth Australian Conference on Information Security and Privacy (ACISP 2000)*, Lecture Notes in Computer Science, Vol. 1841 (pp. 289–297). Springer-Verlag.
- [17] Langford, S. (1995). Threshold DSS signatures without a trusted party. In *Advances in Cryptology, Proceedings of Crypto' 95*, Lecture Notes in Computer Science, Vol. 963 (pp. 397–409). Springer-Verlag.
- [18] Lee, B., Kim, H., & Kim, K. (2001). Strong proxy signature and its applications. In *Proceedings of the 2001 Symposium on Cryptography and Information Security (SCIS2001)*, (pp. 603–608) Osio, Japan.
- [19] Li, C., Hwang, T., & Lee, N. (1995). (t, n) threshold signature scheme based on discrete logarithm. In *Proceedings of Eurocrypt' 94*, Springer-Verlag.
- [20] Malaga, R. (2001). Web-based reputation management systems: Problems and suggested solutions. In *Electronic Commerce Research*, Vol. 1 (pp. 403–417). Kluwer.
- [21] Mambo, M., Usuda, K., & Okamoto, E. (1996). Proxy Signatures for Delegating Signing operation. In *Proceedings of the Third ACM Conference on Computers and Communications Security*, (pp. 48–57).
- [22] Mambo, M., Usuda, K., & Okamoto, E. (1996). Proxy signature: Delegation of the power to sign messages. *IEICE Trans. Fundamentals*, E79-A, 1338–1353.
- [23] Mu, Y., & Varadharajan, V. (2000). Distributed Signcryption. In *Proceedings of INDOCRYPT 2000*, Lecture Notes in Computer Science, Vol. 1977, (pp. 155–164). Springer Verlag.
- [24] NIST. (1995). Secure hash standard. National Institute of Standard and Technology, *Federal Information Processing Standards Publication, 180-1*.
- [25] NIST. (1999). Data encryption standard. National Institute of Standard and Technology, *Federal Information Processing Standards Publication 46-3*.
- [26] NIST. (2000). Digital signature standard. National Institute of Standard and Technology, *Federal Information Processing Standards Publication 186-2*.

- [27] NIST. (2001). Advanced encryption standard. National Institute of Standard and Technology, *Federal Information Processing standards Publication 197*.
- [28] Reiter, M. K. (1996). Distributing trust with the rampart toolkit. *Communications of the ACM*, 39(4), 71–74.
- [29] Sander, T., & Tschudin, C. (1998). Protecting mobile agents against malicious hosts. In *Mobile Agents and Security*, Lecture Notes in Computer Science, Vol. 1419 (pp. 44–60). Springer-Verlag.
- [30] Shamir, A. (1979). How to Share a secret. Technical report, Massachusetts Institute of Technology, MIT/LCS/TM-134.
- [31] Stallings, W. (2000). Network security essentials: Applications and standards. Prentice Hall. ISBN: 0130160938.
- [32] Sun, H., Lee, N., & Hwang, T. (1999). Threshold proxy signatures. *IEE Proc.—Computer & Digital Techniques*, 146(5), 259–263.
- [33] Tzeng, S., Hwang, M., & Yang, C. (2004). An improvement of nonrepudiable threshold proxy signature scheme with known signers. In *Computer & security*, Vol. 23, (pp. 174–178). Elsevier Science.
- [34] Wilhelm, U. (1997). Cryptographically protected objects. Technical report, Ecole Polytechnique Federale de Lausanne, Switzerland.
- [35] Xiong, L., & Liu, L. (2003). A reputation-based trust model for Peer-to-Peer eCommerce communities. In *Proceedings of the IEEE International Conference on E-Commerce (CEC'03)* (pp. 275–284). IEEE Computer Society.
- [36] Yahoo! Auctions, <http://auctions.yahoo.com>.
- [37] Zacharia, G., & Maes, P. (2000). Trust management through reputation mechanisms. *Applied Artificial Intelligence*, 14, 881–908.
- [38] Zhang, K. (1997). Threshold proxy signature schemes. In *Proceedings of the 1997 Information Security Workshop*, Japan, (pp. 191–197).

Omaima Bamasak received her Ph.D. degree from the University of Manchester, UK, in 2006. Her research interests are in designing protocols using cryptography for the provision of security in distributed systems, mobile agent security, electronic/mobile commerce, reputation management, and non-repudiation and fairness protocols.

Ning Zhang received her Ph.D. degree from the University of Kent at Canterbury in 1994, and is now a lecturer in the School of Computer Science at the University of Manchester. Her research interests are in computer security and applied cryptography, e.g., security and privacy in distributed systems, ubiquitous computing, and electronic commerce, with a focus on security protocol design, access control, and trust management.