RDS: Remote Distributed Scheme for Protecting Mobile Agents

Asnat Dadon-Elichai
Department of Computer Science,
Ben-Gurion University,
Beer-Sheva, Israel
+972-8-8655572
asnate@bgumail.bgu.ac.il

ABSTRACT

As of today no solely software-based solution that *a priori* protects the computation of any mobile code and/or mobile agent was presented. Furthermore, Algesheimer *et al.* [1], argue that minimal trust in a third party is essential for the protection of mobile entities. This paper shows that under very mild assumptions, there exists a software-only based solution that can protect any computation of mobile entities in polynomial time bound systems, and without relaying on the minimal trust requirement.

A novel Remote Distributed Scheme, called RDS, is described. RDS is based on fault-tolerant and modest cryptographic techniques and supports an *a priori* protection of any mobile computation that is carried in an honest-but-curious environment ("trusted entities"). We next show, by using on probabilistic techniques, that RDS provides an *a priori* protection for any mobile computation, in any environment, and for any required level of secrecy. We also prove that RDS equivalents, and by thus, provides the same level of protection that is supports by the traditional client/server scheme.

Keywords

Security, mobile agents, mobile codes, secret-sharing, fault-tolerant.

1. Introduction

Mobile computation is a paradigm based on the ability of a service to launch a mobile code or a mobile agent to be executed remotely and act on its behalf. In addition, a mobile agent is an autonomous entity that maintains a status and roams the network under its own control. This paper refers to mobile codes and mobile agents as "agents".

The mobile computation paradigm is attractive, since it supports true parallelism. In addition, it is flexible and can surmount the deficiencies and extends the capabilities of traditional systems, and improves the overall utilization of system resources. Unfortunately, the paradigm also presents new and hard security issues that fall into two main categories: Securing and protecting the hosts from malicious agents and protecting the computation of the agents from malicious hosts. (cf. Figure 1).

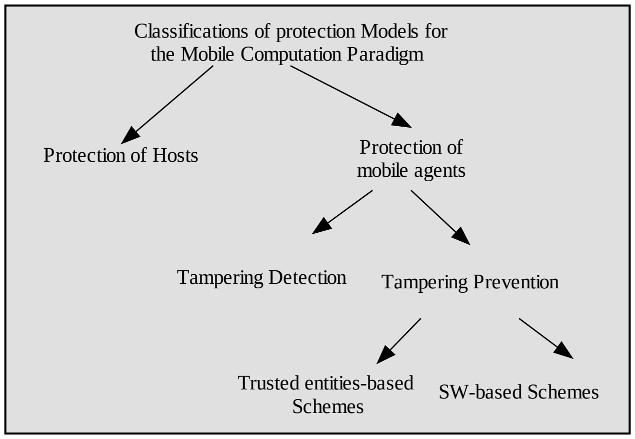
Classifications of protection Models for the Mobile Computation Paradigm
Protection of Hosts
Protection of mobile agents
Tampering Detection
Tampering Prevention
Trusted entities-based Schemes
SW-based Schemes

Figure 1: Classification of Protection Models [14].

The first category aims to protect the status and objects of the hosts by relaying on methodologies such as access control mechanisms, and cryptographic techniques. The second category aims to protect the secrecy and the integrity of the agent's computation. In this paper we focus on the second category.

A malicious host can tamper and improperly influencing the computation of the agent, for example, by manipulating the code and/or intermediate results of the agent, stealing either secrets and/or digital money, or providing inaccurate/faulty data according to its preferences.

The protection of agents is broadly classified into two main approaches: a) *detection* of tempering, that is provided *posteriori* upon the completion of the agent's computation at a given host [2, 4, 7, 12] and b) *prevention* of tempering, that is provided *a priori* during the computation process.

Solutions that belong to the second approach are further classified into two main categories: a) architectures that base on "trusted entities" where an agent can safely carry its computation, or part of it, without being tampered with [1, 2, 4], and b) software-only based solutions. In this paper we focus on the last category that provides an *a priori* protection for any mobile computation in any polynomial bounded environment. Due to readability reasons, we use the term *protection* instead of "*a priori protection*".

Current software-only based solutions are either not fully developed or tend to limit the mobile computation paradigm (e.g. autonomous migration, asynchronous and concurrent computation, and dissemination and assimilation of new information and services). Furthermore, as of today none of the presented software-only based solutions *protect* any computation of mobile agents. Thus, this issue still remains to be solved.

This paper introduces a novel Remote Distributed Scheme, called RDS, that is an only-software based solution for protecting the computation of mobile agents. RDS launches a set of replicated agents, instead of one agent, to carry the computation remotely, and uses the secret sharing technique introduced by Shamir [8] to ensure that each agent leaks as little information as possible to the host that hosts it.

We prove that this scheme protects any mobile computation in honest-but-curious environments. We then show, by relaying on probabilistic techniques, the RDS protects any mobile computation, in any environment and for any required level of secrecy, when some conditions are met.

The paper is organized as follows: Section 2 provides a brief background and discusses related works in the field of an *a priori* protection of mobile agents. Section 3 presents a simple traditional client/server scheme based scenario, describes the behavior of the participants and outlines the implicit assumptions. In Section 4, we present a trusted entity based scheme, demonstrate its behavior and state our desired security conditions. The novel Remote Distributed Scheme is introduced in Section 5. We demonstrate and prove that RDS protects any mobile computation, at any environment and for any level of certainty. In section 6, we compare and discuss the different presented schemes. Conclusions and suggest future directions are provided in Section 7.

2. Related Work

The *protection* of mobile computation by software-only based solutions was deemed as an impossible task [3, 5, 6, 7, 12, 13], until Sanders and Tschudin [10] suggest the Computation with Encrypted Function (CEF) technique that represents the 'mobile code' as a polynomial.

This representation disables the possibly malicious host from tampering with the execution of the mobile code. This approach was further generalized to any arbitrary and polynomial-size depth circuit (function) [4, 11]. It should be noted however that this technique could be used only for the computation of *mobile code*, and only if no information is to leak to the host that hosts it, and only if the originator of the mobile code receives the computation results [1, 4, 10, 11].

This is due to the fact that a malicious host is able to reactivate the encrypted mobile code, for any number of times, until it reveals its functionality and behavior, based on the information that leaks during these reactivated execution. Thus, it is clear that such a requirement is in fact inadequate for a more general task such as a shopping agent.

Algesheimer *et al*. [1] address this issue and state that minimal trust is essential in order to support a non-interactive mobile agent. They suggest generic trusted entity based architecture, in which some of the operations of the agent, are carried out by this trusted entity on behalf of the agent. In their model, different encrypted circuits reside at each of the predefined destinations of the specified route of the mobile agent, where the output of each circuit specify the settings of the proceeding one. Unfortunately, the suggested architecture does not meet the requirements of the mobile computation paradigm, nor it is resilient in case one of the hosts crashes.

In this paper we focused on goals and schemes that are easy to implement and maintain. We think that a software-only based solution that does not limit the mobile computation paradigm is preferable, if it can be provided.

As of today, no solution that does not base on encrypted agents was presented. To best of our knowledge, the RDS, presented in this paper, is the only scheme that relay on modest cryptographic techniques and protects the computation of any mobile agents in polynomial - bounded environments.

3. A Simple Client/Server based Scenario

In this section we present a digital auction scenario that based on traditional client/server scheme, and describe the behavior of the different parties. Based on this scenario, we highlight some of the implicit assumptions, and state the required security goals.

Let us take as an example an automatic, digital public auction scenario, which is carried out by using the traditional client/server scheme. In this example, the client is the auction service ($\mathcal{S}$) of the originator (*O)*, that executes an algorithm ($\mathcal{A}$), and *H* is the auction server. (cf. Figure 2).

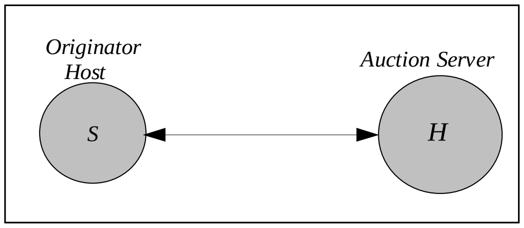

Figure 2: Traditional Client/Server Scheme

Upon invocation, $\mathcal{S}$ opens a connection to *H*, and by thus, informs it about its participation in the auction. This operation also causes *H* to send $\mathcal{S}$ the current best offer. Each time a better offer is submitted to *H*, by one of the other parties that participants in the auction, *H* informs all the participants about the new offer, providing them with the possibility to improve their offers. At any point of the auction, $\mathcal{S}$ may (re)submit an offer. During the computation, $\mathcal{S}$ may also send *H* a special message that includes some address, which is different from its own, informing it to where it should send the winning message in case the offer of $\mathcal{S}$ is the preferred one. The computation of $\mathcal{S}$ is ended either when it closes the connection, or when the auction is ended. When the auction is ended, *H* decides the winner and sends it a message informing it about its winning and encloses the original offer for verification purpose.

In order to disable any party from cheating, we require that all parties will sign the content of their messages by using their private keys. A signed content of message is called a '*transaction*'. This requirement also disables any party from repudiating its offers.

4. Trusted-Entity based scheme

The immediate solution, for protecting any mobile computation, is based on trusted machines or entities. When the service ($\mathcal{S}$) of the originator (*O)* is invoked, it launches its agent ($\mathcal{SA}$), which executes the same algorithm ($\mathcal{A}$), to some predefined trusted host ($H_{Trusted}$), as described in Figure 3.

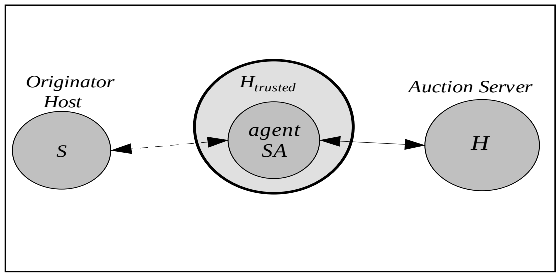

Figure 3: Trusted-machine based architecture

The agent is also provided with a set of transaction (*TS)*. Since $H_{Trusted}$ is a trusted one, it neither tampers with the execution of $\mathcal{SA}$, nor with its results. When $\mathcal{SA}$ relocates on H_{Tusted}, it first opens a connection to the auction server (*H)*. From this point, the computation of the service agent ($\mathcal{SA}$) is carried out in the same manner described in Section 3, until it is ended. By then, $\mathcal{SA}$ sends its status to its originator ($\mathcal{S}$*)*, or to some agreed location.

Lemma 1*:* The computation of the agent in the trusted entity scheme equivalents[1] the computation of the traditional client/server scheme.

The reason that these both schemes provide the same level of protection scheme results from the fact that the agent is relocated on a ***trusted*** host. Furthermore, *H* cannot tamper with either the code or the intermediate results of the agent, since it was not presented with them in the first place.

Clearly the scheme meets the mobile paradigm requirements. Nevertheless, there are several problematic issues to be considered with this scheme. The main concern is that $H_{Trusted}$, is in fact trustworthy, or it is trustworthy but might be *honest-but-curious*; means, it follows the protocol, but tries to infer more information for later on. Secondly, the scheme is clearly not a resilient one and introduces a serious bottleneck issue in the network, and it seems to be inadequate for large systems, and large volume of services and/or agents. Having these arguments, we seek for a robust and resilient solution that leaks as little information as possible to the environment and that can be performed in any environment.

5. Remote Distributed Scheme (RDS)

The Remote Distributed Scheme (RDS) is a novel scheme that base on fault-tolerant and modest cryptographic techniques. It is based on two major ideas: a) replication of agents, and b) sharing the transactions set (*TS)* among these agents, where each agent holds only one share of each transaction.

In RDS, it is required that instead of launching one agent, a set of replicated agents is launched, where all of these agents execute the same algorithm and start with the same initial setting. All of these agents also hold some specific number *id*, which uniquely identifies the underlined set of agents. Each agent communicates only with *H*, and autonomously carries its own computation. Thus producing a star-shape communication pattern. Nevertheless, they are synchronized, as a result of *H*. (cf. Figure 4).

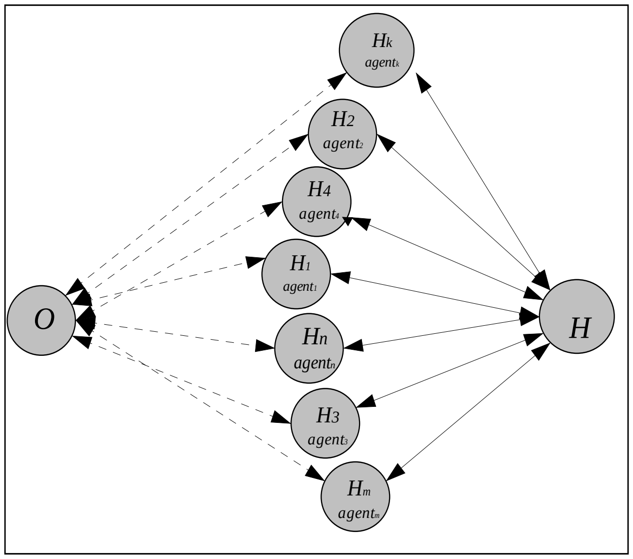

Figure 4: Remote Distributed Scheme

Each of these agents holds the same *id*, which uniquely identifies the set of agents it belongs to, and a set of transactions' shares (*TS*), one share of each transaction. Each of these shares is signed by using the private key of the originator. We will elaborate on these shares later on.

When a set of agents is launched, each agent relocates itself on a different host, and opens a connection to *H*, and provides it with a number *id*. Based on the messages each agent receives from *H*, its algorithm ($\mathcal{A}$), and its inner setting, the agent, updates its setting, carries its computation and autonomously chooses one share from *TS*, and sends it to *H*. Having enough shares of the same transaction, *H* can uniquely construct the transaction, performs its computation, and broadcasts its response to all the agents that provide it with the same *id* number. We emphasize once again, that there is no communication between the agents themselves, and by thus between their hosts, and the only allowed communication is between the agents and *H*. This process is continued until the computation is ended. At the end of the computation, each agent sends its status to its originator or to some predefined location.

[1] Refer to [9].

The reminder of the section is organized as follows: first, we provide some assumptions and explanations. Next, we formalize the construction and the distribution of the transactions shares among the agents. Next, we prove that the RDS supports the same level of protection that is provided by the client/server scheme. We finalize this section with an extended example of a RDS based scenario.

5.1 Assumptions

Assumption 1: All the hosts that participate in any computation are polynomial time bounded.

Assumption 2: All the hosts use a public key cryptosystem, and each host signs all its transactions by using its private key and every party can authenticates any transaction.

This assumption assures that cryptographic techniques can be used without considering cases where hosts can forge the signature of other hosts.

Assumption 3: A problem *P* can be solved by an algorithm $\mathcal{A}$, and a predefined set of transactions *TS*.

By this assumption we actually decouple between the algorithm that implements some function (circuit) and the transactions that specify the setting for each round of the computation.

Assumption 4: All the hosts that host the agents that participate in some computation do not communicate with each other.

This assumption is made to exclude the case of an advisor (oracle) attack; an entity that correlates a collusion attack.

5.2 Construction of Shares and Transactions

Based on *Assumption 3*, in order to reveal the goal of an agent, we need to have its algorithm and inner settings. Since the algorithm $\mathcal{A}$ is known to all the hosts, H_l, $l = 1\ldots n$, that host the agents, there is a need to distribute the transactions set among the agents, and by thus the hosts, in such a way that:

a) Enables *H* to relate only to authentic shares. The reason behind the this requirement comes from the fact that due to the communication latency or even due to some improper actions of one of the hosts, *H* might hold shares of different transactions.

b) Enables *H* to easily and uniquely determines the shares of the relevant transaction.

c) Enables *H* to easily resolve the transaction, having enough shares.

d) Leaks as little information as possible to the hosts themselves.

e) Disables a malicious host, from improperly influencing the computation by sending all the transactions to *H*.

The *(k, n)*-secret sharing threshold scheme presented by Shamir [8], is used to construct the shares. Let $TS = \{t_1 \ldots t_m\}$ be a predefined set of signed transactions, based on *Assumption 2*, each $t \in TS$ is formed as (t, γ) where γ is the legal signature of *O* for *t*.

For each $t_i \in TS$, we define the set $\{t_{i1} \ldots t_{in}\}$ to be the *n* shares of t_i, that were produced by using the *(k, n)* secret sharing threshold. Let $st_{ij} = ((i, t_{ij}), \gamma)$, where γ is a legal signature of *O*, (the originator of the agents), on (i, t_{ij}).

The honest dealer, *O* (i.e. the originator), computes off-line the set $\{st_{ik}\}$, $i = 1\ldots m$, $j = 1\ldots n$ and provides the j^{th} agent with the set $\{st_{1j} \ldots st_{mj}\}$. (It's shares of all the secrets).

Note that each share is signed and holds the index of the transaction it relates to. This construction enables *H* to easily authenticate each share, and to easily determine all the relevant shares of the same a transaction within one round of the message collection phase. Having at least *k* authentic shares, of the same secret (transaction), enables one to easily resolve the transaction.

The only issue remains to be solved is how to decide the value of *k*, in such a way that a correct transaction can be uniquely resolved. Recall that the *(k, n)* secret-sharing threshold scheme, described in [8], only requires that at least *k* shares of the same secret are provided, while assuming that all the shares are authentic, and not tampered with. Clearly, choosing $k = \lceil (n+1)/2 \rceil$ resolves this issue.

Now, *H* flushes all the shares where their first component does not equal the majority, and at the end of this phase, it holds a set of authentic shares $\{(i, D_{ij})\}$ where $|\{(i, D_{ij})\}| \geq k$.

5.2.1 *Secret Sharing Threshold Scheme - Overview*

A secret *X* is divided into *n* shares $\{X_1 \ldots X_n\}$ in such a way that:

- Knowledge of *k* or more X_i shares makes *X* easily computed.
- Knowledge of less then *k* shares leaves *X* completely undetermined (in the sense that all possible values are equally alike).

A random polynomial *k*-degree function *f(x)* over $\mathbb{Z}_p$ is chosen, where $f(0)=X$, and $f(i)= X_i$, $\forall i$. Any *k* or more values from $\{X_i\}$ enable the resolution of *f(x)* and the free component *X*. For more details about the *(k, n)* secret sharing threshold scheme, please refer to [8].

5.3 The Correctness of RDS

In this section we prove that RDS supports the same level of secrecy as provided by the client/server scheme. We first prove this claim for honest-but-curious environments, by then we prove that it holds for any environment.

5.3.1 *Protecting Mobile Computation in "Good" Environment*

We start by providing a definition of what is a good environment. This definition is valid for the appropriate values of *k* and *n*, as described in Section 5.2. Note that a good environment is not a one that requires that all the hosts that host the agents are trustworthy or honest-but-curious, but only *k* of them. This actually implies that the correctness of RDS is not violated, even if *n-k* of the agents are relocated on *H* itself. In the next section, we show that this assumption can be eased.

Definition 5: An environment is *good* if for any given set of *n* hosts, that participate in some computation, at least *k* hosts are either trustworthy or honest-but-curious.

A mobile computation in RDS requires that a set of *n* agents be launched instead of one agent. Each of the agents relocates itself to some host $\in\{H_1 \ldots H_n\}$ where H_i is a neighbor of *H*. *H* presents the same execution described in Section 3 and Section 4, with one major difference: instead of receiving one message at each round of the computation, it collects at least *k* authentic shares of the same transaction and resolves it before continuing with its execution. This computation continues exactly as described above until it ended. At the end of the computation, all the agents send their status to their originator or to some predefined location. We make the following claims:

***Claim 2*:** The computation RDS equivalents[1] the computation of the traditional client/server scheme.

Proof: All the agents that participate in some computation execute the exact same algorithm $\mathcal{A}$, and start with the same initial settings. Based on *Assumption 2,* at least *k* of these hosts are trustworthy or honest-but-curious. Thus, the executions of at least k agents are authentic, and at least *k* agents autonomously compute $\mathcal{A}$ properly, providing *H* with at least *k* shares of the same exact transaction.

Having a set of at least *k* authentic shares of the same exact transaction enables *H* to easily determine the relevant ones and to easily construct the transaction correctly, based on the *(k, n)* secret sharing technique. Since the transactions are signed by using the originator private key, no host can forge them, including *H*.

Now, we need to show that for a given *P* and algorithm $\mathcal{A}$, the RDS and the client/server produce the same stream of transactions. Since the proof is simple but a tedious one we omit it.

Now, since in both schemes, all the agents execute the same $\mathcal{A}$, and start with the same initial setting, and since at least *k* of them produce the same stream of transactions, based on the same results from *H*, then the computation of RDS equivalents the computation of the client/server scheme.

Theorem 3: RDS provides the same *protection* that is provided by the client/server scheme.

Proof: Results immediately from *Claim 2*.

The remaining interesting issue for RDS is, whether or not, at least one host is able to impede the computation. This issue is important since neither the algorithm nor the agents' statuses are concealed from the hosts that host them.

Based on *Definition 5,* and by selecting *k* to be $\lceil (n+1)/2 \rceil$, it is clear even if *n-k* of the hosts are malicious, and/or even cooperate with one another, the computation cannot be forged. This is also true if *n-k* of the agents reside in *H* itself.

Furthermore, at the end of the computation, *O* receives at least *k* authentic results that include all the signed responses of *H*, and the corresponding shares. These results enable one to integrate a *posteriori* analyzing system that can trace the computation evaluation, and identify the malicious host, if there was one, and the exact point where the computation was tampered with.

5.3.2 The Correctness of RDS in any Environment

We showed in Section 5.3.1 that RDS protects any mobile computation in a "good" environment. Unfortunately, this requirement seems to be somewhat problematic due to the fact that malicious entities do not usually declare themselves as such.

So, we ask ourselves under what conditions RDS can protect any mobile computation in any environment, thus, easing the need of *Definition 5,* or in other words ask the following question: "*If there is no knowledge about the trustworthiness of the hosts that host the agents, can the computation of RDS can be protected for any required certainty factor?"* We answer this question affirmatively by proving the following claim:

Claim 4: For any certainty factor γ, exists RDS for which the probability of a "faulty computation" is smaller then γ.

Proof: Let us assume that the RDS does not protect the computation of agents that implement some algorithm $\mathcal{B}$. Clearly, a false computation occurs when *H* can present at least one authentic transaction, t_f that was not constructed by any *k* set of the agents. We now examine the different strategies that might be used to produce this faulty transaction.

1. Based on *Assumption 2,* neither *H* nor any of the other hosts can autonomously produce t_f, since it signed by the private key of the originator *O*.

2. Thus, at least *k* hosts, sends *H* the proper shares, enabling it to successfully construct t_f. Based on *Assumption 4,* the hosts do not communicate with each other, and by thus cannot cooperatively decide on some common strategy to forge the computation.

3. This mean that at least *k* hosts autonomously and mutually either chose some random value and send it to *H* enabling it to construct t_f, where the probability of such a case clearly zero (0) based on *Assumption 1*, and *Assumption 2;* or

4. At least *k* hosts autonomously and uniformly select the share of the same exact transaction t_f, from *TS,* instead of sending it the shares of the correct transaction t_s. Assuming the existence of *m* different transactions, and *n* <*2k* hosts, then the probability of having such a situation equals to:

$$P[t_{fs}] = \frac{1}{m^k}\binom{n}{k}^{n\le 2k} < \frac{1}{m^k}\frac{(2k)!}{k!k!} = \frac{1}{m^k}\frac{2k(2k-1)\ldots(k+1)}{k!} =$$

$$= \frac{1}{m^k k!}\prod_{i=1}^{k}(k+i) = \frac{1}{k!}\prod_{i=1}^{k}\frac{k+i}{m}$$

Choosing $ck \le m$ defines the probability of such a situation to be:

$$P[t_f \neq t_s] = \frac{1}{k!}\prod_{i=1}^{k}\frac{k+i}{m} \overset{ck<m}{\leq} \frac{1}{k!}\prod_{i=1}^{k}\left(\frac{k+i}{ck}\right) = $$
$$= \frac{1}{c^k k!}\prod_{i=1}^{k}\left(1+\frac{i}{k}\right) \leq \frac{1}{k!}\left(\frac{2}{c}\right)^k$$

Thus, setting the relation between *m* and *k* to be a large constant *c*, defines the probability of such a situation to be as small as required, and by thus defines the required certainty factor γ to be as large as required. This means that RDS protects the computation of any mobile computation in any environment.

5.4 An Extended example of RDS based Scenario

Before we demonstrate the example, we start with some notations. A message is formed as follows: *{X, Y, Id, Payload}* where: *X* defines the message source, *Y* defines the message target, *Id* that uniquely identifies a set of agents, and *payload* defines a transaction. A transaction is formed as follows: {(*Type, Content), Value*}, where: *Type* defines the type of the *Content,* and *Value* is the legal signature of the party that initiates the message. The *Content* can be a simple numerical value or some package deal. The type can be one of the following:

- *suggestOffer:* used by the agents to submit an offer to *H*.
- *sendWinningTo*: used by the agents to inform *H* to where to it should send the winning notification.
- *YouWin*: used by the host *H* to inform the winner about its winning.
- *CurrentBestOffer:* used by the host *H* to inform all the auction participants about the current best offer.

Thus, the Transaction Set (TS), of the originator O includes *m* transactions that look like:

$\{t_1=[(suggestOffer, Offer_1), Sig_1]$
…
$t_k = [(sendWinningTo, Address), sig_k]$
…
$t_m = [(suggestOffer, Offer_m), Sig_m]\}$

By then the (k, n) secret sharing is used to produce the following set of shares: $\{\{t_{11}\dots t_{1n}\}, \dots\{t_{m1}\dots t_{mn}\}\}$. From these shares we produce the following set of shares:

$\{((1, t_{11}), sig_{11}))\dots((1, t_{1n}), sig_{1n}))$
…
$((m, t_{m1}), sig_{m1}))\dots((m, t_{mn}), sig_{mn}))\}$

Assume that a service of the originator *O* launches a set of *n* agents in order to participate in an automatic public auction. Each of the agents behaves as described in Section 5.3, and holds a set of *m* shares. The J^th agent is provided with all the shares appear in the j^th column. After the agents are relocated, each opens a connection to the auction server *H*, informing it that they participate in the auction. The auction server *H* is responding with a message: *((CurrentBestOffer,Offer), $H_{signature}$).*

By receiving this response from *H*, all the agents perform their algorithm and update their new setting. Since all the agents execute the same algorithm and have the same setting and received the same result, all of the agents make the same decision. They either terminate the computation since the presented offer exceeds their best one, or they can select the shares of some transaction and submit it to *H*, or they can send a transaction that informs H to where to send the winning notification.

H uses the *id*, to determine the shares of the same group of agents, and uses the index provided with each share to determine which shares relate to the same transaction. It resolves the transaction, and evaluates all the offers it receives, it chooses its preferred one and informs all the participants about this offer.

When the auction is ended, *H* sends the message *(youWin, (Verification, details), $H_{signature}$)* to the winner, where: *Verification* is actually the authentic signed offer of the agents *(suggestOffer, Offer), OfferSig)*, that is provided as evidence, and *details* that is used for any purpose such as payment terms, etc. The $H_{signature}$ is the legal signature of *H* on the contract.

6. Schemes Comparison and Discussion

Due to lack of space, we briefly raise and discuss different considerations in relation to the performance, the communication overhead and availability of the above presented schemes; namely the traditional client /server scheme (*CSS*), the trust entity scheme (*TES)* scheme, and the Remote Distributed Scheme (*RDS*). It should be noted however, that TES is a private case of RDS, where the launched set of agents includes exactly only one agent, and each of its shares is the transaction itself. Furthermore, CSS is an instance of TES, where the trusted host is actually the originating one. (cf. Figure 5).

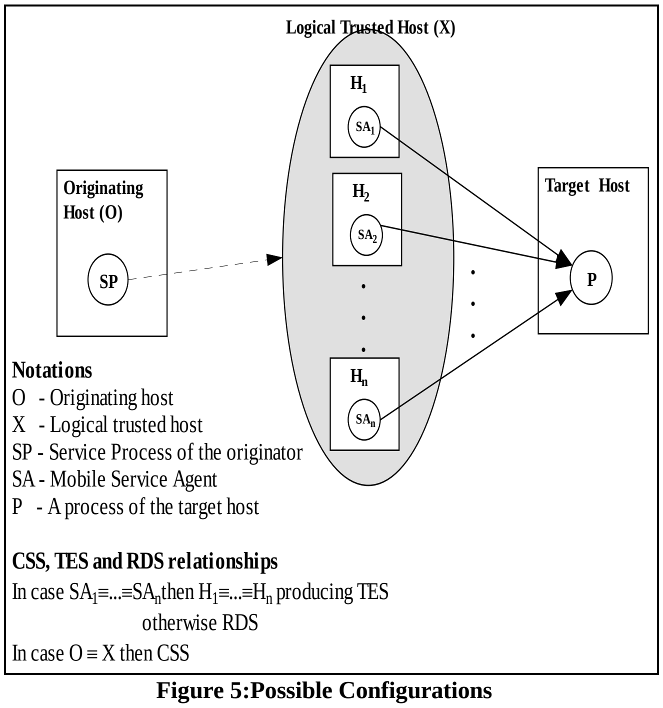
Logical Trusted Host (X)
Originating Host (O)
SP
H1
SA1
H2
SA2
Hn
SAn
Target Host
P
Notations
O - Originating host
X - Logical trusted host
SP - Service Process of the originator
SA - Mobile Service Agent
P - A process of the target host
CSS, TES and RDS relationships
In case SA1≡...≡SAn then H1≡...≡Hn producing TES
otherwise RDS
In case O ≡ X then CSS

Figure 5:Possible Configurations

Communication Overhead. Clearly, the intrinsic communication overhead of RDS is higher than a one that base on either CSS or TES. While the communication overhead of CSS is affected from the exact number of transactions sent by the originating host and the volume of the corresponding results, the communication overhead of both TES and RDS composed of four factors:

a) Launching a set of agents, possibly one agent, at the beginning of the computation to perform the mobile computation remotely.

b) Receiving and evaluating at least $k \leq n$ replicated results, at the end of the computation.

c) Sending one transaction to the target host and receiving its response by all agents. In RDS case, each of the *n* agents holds all shares of all the transactions. Based on [8], the size of each share equals to the original size of the secret. This results a cost of *2n* messages for sending *n* shares of the same transaction and receiving the result of the target host by all agents.

d) The number of transactions sent to the target hosts from all agents during the computation. Note, that this number might differ greatly from the number of transactions included in the transactions set, $|TS|$.

In addition, there are several issues that apply only on TES and not on RDS. Trusted hosts are machines that need to be trusted by all other hosts in the network. This results that these Such a requirement implies the possibilities of inadequacies due to strict maintenance and support, causing two possible effects: a) producing a bottleneck in the network, and b) might be inadequate for large networks, large volume of agents and large variety of services that are required to meet the mobile computation needs.

As a result it is clear that trusted entities are very specific and there are few such hosts in the network. Based on this observation, it can be assumed that a trusted host is neither a neighbor of either the originating host or the target host. Furthermore, it is reasonable to assume that there are cases where the distance between the trusted host and the target host is greater than the distance between the originating and the target ones.

Performance. On the three schemes there are a communication delay. While the communication latency of CSS and TES might be affected from available bandwidth, congested lines, on the route between the originating host and the target host, the latency of RDS is minimal. This is due to the fact, that all the agents relocate themselves on the neighbors of the target host.

Nevertheless, the performance of RDS also affected from the fact that the target host cannot resolve a transaction until it collects at least *k* shares of it. This results that the median slowest agent; rather then the most slow one, determines the communication latency of each round of the computation. Furthermore, this means that each time a transaction is sent by at least *k* agents, the median slowest agent of these agents determines the latency. Thus, the replication of the agents may actually improve the performance of the computation by assuring that slow/faulty hosts do not impede the progress of the computation.

Resiliency. RDS is clearly a resilient and tolerant scheme in contrast to CSS and TES. While a computation of the last two schemes might terminate due to any crash of some host or connection along the route between the originating or the trusted hosts, and the target host, RDS is tolerant to *n-k* such events.

7. Conclusion and Future Work

In this paper we studied the issue of an *a priori* protection of any mobile computation in possibly hostile environments, and focused on goals and schemes that are easy to implement and deployed. In this study, we did not require an absolute secure solution, but looked for a one that supports the same level of protection that is provided by the traditional client/server scheme.

We introduce a novel Remote Distributed Scheme, called RDS, that is a general instant of the traditional client/server scheme. RDS is based on two main ideas: a) replication of agents, and b) sharing the transactions among these agents by using the *(k, n)* secret-sharing threshold scheme.

We showed that RDS supports this requirement in any environment, while not restricting any of the other requirements introduced by the mobile computation paradigm. RDS assures that the information gained by the hosts that host the agents is no more than the information gained by the traditional client/server one. The scheme also protects the computation from faulty and erroneous behavior of malicious hosts, and enables the detection of malicious host that participates in the computation. Furthermore, it is easy to implement and to maintain since it is executed in a clear-text mode. None of the current solely based software solutions supports all these abilities.

RDS provides all these abilities if at least *k* hosts are *known* to be either trustworthy or honest-but-curious. In case this information is unavailable, we provide a well-specified estimation for protecting the computation by defining the relation between the number of the transactions and the number of the participant hosts. By using this estimation we prove that the computation is protected in any environment, and for any required certainty factor.

RDS also benefits from the ability to tune the level of the required security, by either increasing /decreasing the number of agents based on the trustworthiness or hostility of the network. On one hand, in case the network appeared to be trustworthy, the number of the agents can be reduced. On the other hand, in case the hosts in network appeared to be distrusted and/or malicious then the number of the agents can be increased to provide the required certainty factor.

Current investigation is focused both on theoretic and implementation directions. We are currently implementing RDS for analyzing its feasibility for different problems. On the theoretic side we study issues that relate to the communication/security tradeoff in mobile computation paradigm.

8. REFERENCES

[1] Algesheimer J., Cachin C., Camenisch J., and Karjoth G. Cryptographic Security for Mobile Code. IEEE Symposium on Security and Privacy, 2001.

[2] Bennet, Y.S. A Sanctuary for Mobile Agents. Secure Internet Programming, LNCS, Vol. 1603, 1999, 261-273.

[3] Barak, B., Goldreich, O., Impagaliazzo, R., Rudich, S., Sahai, A., Vadhan, S., and Yang K. On the (im)possibility of obfuscating programs. LNCS, Vol. 2139, 2001, 1-18.

[4] Cachin, C., Camenisch, J., Kilian, J., and Muller, J. One-Round Secure Computation and Secure Autonomous Mobile

Agents. Automata, Languages and Programming, 2000, 512-523.

[5] Hohl, F. A Model of Attacks of Malicious Hosts Against Mobile Agents. 4th Workshop on Mobile Object Systems: Secure Internet Mobile, 1998, 105-120.

[6] Hohl, F. Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts. Mobile Agents and Security, LNCS, Vol. 1402, 1998, 92-113.

[7] Giovanni, V. Cryptographic Traces for Mobile Agents. Mobile Agents and Security, LNCS, Vol. 1419, 1998, 137-153.

[8] Shamir, A., How to Share a Secret. CACM, Vol.22, 1979, 612-613.

[9] Sipser, M. Introduction to the Theory of Computation. PWS publishing company, Boston MA, 1997.

[10] Sander, T., and Tschudin, F. Protecting Mobile Agents Against Malicious Hosts. Mobile Agent Security, LNCS, Vol. 1419, 1998, 44-60.

[11] Sander, T., Young, A., and Yung, M. Non-Interactive CryptoComputing For NC^1, Proceedings 40th IEEE Symposium on Foundations of Computer Science (FOCS), 1999, 554-567.

[12] Xun, Y., Xiao, W., Lam, F., and Kwok, Y. A Secure Intelligent Trade Agent System. LNCS, Vol.1402, 1998, 218-228.

[13] Minsky, Y., van Renesse, R., Schneider, F.B., and Stoller, S.D. Cryptographic support for fault-tolerant distributed computing, Proceedings of the Seventh ACM SIGOPS European Workshop, Connemara, Ireland, September 1996, 109-114.

[14] Kotzanikolaou, P., Burmester, M., and Chrissikopoulos, V. Secure Transactions with Mobile Agents in Hostile Environments. LNCS, Vol. 1841, 2000, 289-297.