Hindawi
Security and Communication Networks
Volume 2017, Article ID 3606424, 18 pages
https://doi.org/10.1155/2017/3606424

Research Article

BROSMAP: A Novel Broadcast Based Secure Mobile Agent Protocol for Distributed Service Applications

Dina Shehada,[1] Chan Yeob Yeun,[1] M. Jamal Zemerly,[1] Mahmoud Al Qutayri,[1] Yousof Al Hammadi,[1] Ernesto Damiani,[1] and Jiankun Hu[2]

[1] Electrical and Computer Engineering Department, Khalifa University, Abu Dhabi, UAE
[2] School of Engineering and Information Technology, The University of New South Wales, Sydney, NSW, Australia

Correspondence should be addressed to Chan Yeob Yeun; cyeun@kustar.ac.ae

Received 8 September 2016; Revised 26 January 2017; Accepted 8 February 2017; Published 29 March 2017

Academic Editor: Qing Yang

Mobile agents are smart programs that migrate from one platform to another to perform the user task. Mobile agents offer flexibility and performance enhancements to systems and service real-time applications. However, security in mobile agent systems is a great concern. In this paper, we propose a novel Broadcast based Secure Mobile Agent Protocol (BROSMAP) for distributed service applications that provides mutual authentication, authorization, accountability, nonrepudiation, integrity, and confidentiality. The proposed system also provides protection from man in the middle, replay, repudiation, and modification attacks. We proved the efficiency of the proposed protocol through formal verification with Scyther verification tool.

1. Introduction

A mobile agent (MA) is intelligent software that has the ability to migrate from one node to another, performing tasks on behalf of its owner [1, 2]. In agent-based systems, the nodes where agents migrate, execute, and collect data are called platforms. MAs reduce the load on the network and require low bandwidth. MAs also add scalability to the network; that is, it is easy to add new participating agents to the system. They have the ability to perform asynchronous tasks. Moreover, MAs are small and operate based on the idea of moving computations to data rather than moving data to computations [3–6], reducing the number of computations to be performed on owners. Owners dispatch their agents carrying the request, and agents execute on other service platforms, doing most of the work by taking the advantage of their computational power and eventually coming back with useful information. All these features make them suitable for real-time information retrieval applications with distributed databases that are otherwise time-consuming and operationally costly to search [7, 8]. An example of real-time distributed applications is Location-Based Service (LBS) ones. LBS applications provide users with geospecific services and information based on their location and preferences. LBSs are used in different contexts, such as health, entertainment, travel, and many others [9, 10]. The use of LBS applications provides users with the needed information and saves their time, effort, and money. In LBSs, a large amount of information is stored in different places and the challenge lies in ensuring fast and efficient information retrieval process upon the user request [11]. MAs are small, fast, autonomous, have a distributed architecture, and have the ability to migrate in the network to search for information. Therefore, incorporating mobile agent system (MAS) with LBS will enhance the system performance and efficiency.

Although agents offer many practical features to distributed systems, MAs face classic security breaches as any other software that is distributed in an open network, hence ensuring their security raises challenges [12]. The flexibility and mobility of agents in the open network increase their vulnerability to different security attacks such as masquerade, modification, man in the middle (MITM), replay, repudiation, eavesdropping, and denial of service (DOS) attacks [3, 13–15]. If these vulnerabilities are not properly addressed, they will not only reduce the performance and integrity of the system and services provided,but will also put user privacy at risk. In addition to their locations, users of LBS applications might need to reveal some sensitive information. Therefore, it is important for application users to trust that their data are not going to be exposed to any untrusted entity.

To overcome such security problems, a novel BROadcast based Secure Mobile Agent Protocol (BROSMAP) for distributed service applications such as LBS is proposed. BROSMAP provides significant improvements over existing protocols found in the literature. While previous research efforts could address only small subsets of the security properties identified as important for MASs, BROSMAP's innovative design provides security requirements coverage while staying lean enough to guarantee performance. It provides a generic solution for different security threats. It also provides the fundamental security requirements for distributed service applications. The proposed broadcast dispatch mechanism in addition to the use of hybrid approach of symmetric and asymmetric cryptography methods helps the system in maintaining a good performance compared to other protocols in the literature. In addition, BROSMAP has the flexibility of staying operational, even if one of its agents is maliciously killed or lost.

The remaining part of the paper is structured as follows. In the next section, requirements of secure MAS are discussed. Section 3 introduces the related works. In Section 4, details of the novel BROSMAP are discussed. Section 5 provides detailed security analysis in addition to analysis of results of formal verification proof carried with Scyther tool. Section 6 discusses the performance evaluation of BROSMAP compared to other similar protocols. Section 7 shows details of the implementation the BROSMAP based LBS application carried out with Java Agent Development Environment (JADE) [16, 17]. Finally, Section 8 summarizes the contributions and highlights future work.

2. Security Requirements for MASs

As noted in the previous section, MASs are vulnerable to different types of attacks. Such vulnerabilities do not only affect users' security and privacy but also the system productivity. The security research community has identified some key security requirements for MASs; the following security requirements should be satisfied [13, 18–21]:

(i) Anonymity: privacy of the user (owner platform) is maintained through hiding its identity from being exposed.

(ii) Authorization: agents attributes should be verified and accordingly are either granted or denied the access to platform resources.

(iii) Confidentiality: user information, data, or any other sensitive information should remain secret and not be exposed to unauthorized platforms or agents.

(iv) Integrity: user information, data, and other sensitive information should be protected from unauthorized changes by a malicious party and any unauthorized change should be detected.

(v) Mutual authentication: communicating platforms should verify each other's identity. This is usually done by using Public Key Infrastructure (PKI) where each platform identifies itself by its certificate or signatures.

(vi) Availability: platforms should be always available and ready to provide services. In addition, communication should not be disrupted at any time.

(vii) Nonrepudiation: actions should be traceable to their actors. Any platform cannot deny its results or actions.

(viii) Accountability: attackers or malicious platforms are held responsible for their malicious actions.

In the next section, a review on some of the related work proposed to secure MAS is presented.

3. Related Works

Securing MASs has been addressed by many researchers. Some proposed works address the issue of user privacy and providing anonymity to system users [22–24]. This feature is important if the application involves dealing with sensitive information. Others focus on providing a scheme to protect the dynamic itinerary of the MA or propose authorization schemes to protect platforms from malicious agents [25–27]. Some authors propose a solution to repudiation and replay attacks, providing the possibility of legitimate reexecution of agents [28], while others propose a new MAS architecture to protect the system from losing all its data in case the MA is killed [29]. Other authors put forward a cryptographic and/or trusted platforms based approaches to provide secure MASs [30–34].

In [22], user (owner platform) anonymity and secrecy of path traversed by the MA are addressed. Intermediate helper nodes called mixers are visited by the MAs on their itineraries. Visited mixers in the MA journey store their addresses in the agent state. The MA carries on, randomly migrating from one node to another. At the last itinerary, the addresses of the visited mixers that are stored in the MA state are used to direct the agent to the owner platform. The protocol provides anonymity to the MA owner and to the path of agent. Confidentiality and authorization of data access are ensured through encryption. Authentication of users is not provided and it is assumed that exchanged messages cannot be intercepted and that they all come from legitimate users. A drawback of the proposed protocol is the inability to recover from the malicious killing of an agent. If an agent is lost, the collected data are lost and a new MA has to be resent again. Furthermore, the proposed system introduces random delays, due to the random decisions of the mixers. Therefore, this approach is not suitable for many applications.

An anonymous authentication protocol is proposed in [23]. MAs exchange messages that involve information required to authenticate the user. User information is kept confidential through symmetric encryption. A freshly generated nonce and a secret key that are known to the user agent only are needed for a successful decryption of the message. Mutual authentication between interacting agents and protection from replay, MITM, and masquerade attacks are claimed.

The authors in [24] proposed another anonymity scheme based on the idea of a last input first output (LIFO) buffer of data that the agent carries with it throughout its journey. Each platform generates its own secret key, a hash of the previous, current, and next destination, and a fresh nonce. The platform encrypts the nonce and the hash along with the previous destination with its secret key and pushes the hash into the stack. When the agent finishes its journey, each platform uses its secret key to decrypt the information available on the stack and sends the agent to its previous destination. The stored hash is used to check the integrity of the path stored to detect if the stack was tampered. Therefore, only the first platform knows the actual address of the owner platform. At the starting point, the LIFO buffer is filled with random garbage data to prevent platforms from reconstructing the address of the owner platform. A weakness of this approach is the size of garbage that the buffer should be filled with. It should be randomly chosen in a way that is big enough to protect the owner platform identity and provide protection from analysis attacks. At the same time, it should also be small enough to fit the MA limited memory size, taking into consideration other itinerary information to be stored later. The protocol also assumes that all MAs in the network have the same code and state, which is an unrealistic assumption for MASs [22].

The authors in [25] proposed a cryptographic and trusted platforms based protocol, which ensures the secrecy of the dynamic itinerary generated at run time. Trusted platforms are the main key here; they collect information of other platforms securely and decide the agent's next itinerary. The itinerary information is encrypted and can only be accessed by the intended platform. In addition to providing secrecy of agent itinerary, the protocol also provides integrity to agent code through hash and asymmetric encryption. Mutual authentication between MAs and platforms is provided. The protocol best suits systems which require time-consuming tasks; otherwise, the system introduces a timing overhead that is not suitable for many applications [28, 35, 36].

A protection protocol preventing illegitimate reexecutions and replay attacks of MAs is proposed in [28]. An authorization entity issues a unique identifier called trip marker for each MA. Agents hold counter values which indicate the number of times an agent has been executed. Sometimes an MA needs to execute more than once on the same platform. The counter makes that possible without the need to generate a new trip marker for the agent. Tracks of trip markers and counter values of executed agents are kept on each platform. When the agent requests to be executed, platform checks if the carried trip marker is already in its database. If there is no match then it saves the agent information and executes it. If a match is found, it checks the value of the counter carried by the agent and the value it has in its database; if the counter is incremented, then this is a legitimate reexecution; otherwise, it detects a replay attack and kills the agent. The protocol is verified through simulating replay attacks with Java Agent Development Environment (JADE) [16, 17]. The results verified the ability of the proposed protocol in detecting all simulated replay attacks; however, the protocol is still vulnerable to other attacks such as eavesdropping and repudiation attacks. Although the proposed protocol provides mutual authentication and authorization, it does not support agent cloning as each copy of the agent should carry a new trip marker. In addition, malicious killing of the MAS results in the loss of its collected data [35].

To protect and manage information authorization, a Visa Based Authentication Scheme (VBAS) for MASs is proposed by [26]. The user generates an agent and grants it a passport containing the agent information, that is, name of its owner, identity, time stamp, certificate, and other data required for agent authentication. The passport is sent to an authentication server (AS) which verifies the passport and issues a Visa that includes all the resources the MA granted access for. The Visa is verified by a verification server and sent back to the authentication server for final verification, before sending it to the agent. With the Visa issued by the trusted Visa and authentication servers, platforms are able to detect if a malicious agent is unauthorized to access some resources. The protocol provides an integrity check for the passport and Visa. The protocol assumes the availability of a secure communication channel. However, if the channel is compromised, then the passport and Visa information sent in clear text are disclosed. Although the Visa and passport might not contain sensitive information, sending them in clear text makes the system vulnerable to the denial of service attack, where attackers might maliciously tamper with the passport and Visa of an agent and send the tampered version to overwhelm legitimate platforms [37].

Another authorization protocol for MAS is proposed by [27]. The work provides an extension to [38] protocol where malicious identification police (MIP) scan MAs for malicious code. MIP use a database containing malicious codes previously detected; if MIP find that the agent is genuine, then the agent is executed; otherwise, the agent is killed. The extension proposed in [27] allows platforms to scan the agent's code for platform specific policy file. The file includes privileges and permissions granted to agents. The agent's code is scanned and compared to the policy file of each executing platform. The database contains malicious codes that are known to the system; however, agents with unknown codes are executed and kept under monitoring. If the agent starts to act maliciously, then its code is eventually added to the database. This delay might jeopardize system security. Moreover, any change made to the policy files needs to be reported to the Attack Identification Scanner (AIS) [39]. To support integrity of agent code against malicious platforms, another protocol is proposed in the paper. The protocol is an improvement to the classic Root Canal protocol [40], which was proven to be vulnerable to repudiation attacks. By signing the hash of the code, The eXtended Root Canal (XRC) protocol provides a proof for detecting malicious platforms and, therefore, protection from repudiation attacks as well [34].

Authors in [41] proposed a security framework for mobile agent platforms. The proposed framework introduces the idea of multiple levels of access control of platforms resources for MAs. Agent level of access is dependent on their history of traversed itineraries. Each platform visited by the MA signs it and then decides its next destination. Platforms check the history of visited platforms and based on the list it decides how trusted is the MA and grant it access accordingly. The framework provides one-way authentication and authorization. However, the disadvantage of the proposed framework lies in the increase of the path history and signature chain as the MA migrates from platform to another. Another disadvantage is the assumption that platforms are able to identify malicious platforms in the system.

A matrix hop architecture for MASs is proposed by [29]. The architecture protects the system from losing its functionality by a malicious killing or loss of MAs. Data generated at every platform are sent back to the owner platform before the migration of the MA to the next platform. Moreover, the architecture enables the system to recover if the agent is killed. The owner platform keeps track of data sent by platforms; therefore, if the agent is killed, then the owner will send the agent to the next destination platform, to carry on its journey.

A secure MA information transfer system for e-health applications is proposed in [30]. Information needed for decryption is only provided upon the verification of platforms legitimacy. First, the owner sends its agent for verification along with a token and encrypted data. Upon verification of the agent and the integrity of the code, the receiver sends the token with some other information proving its legitimacy to the sender. Then the sender sends back a one-time secret key required to decrypt the data. Mutual authentication between sender and receiver and data confidentiality are provided. Moreover, data are only accessible to authorized and verified parties [42, 43].

Confidentiality, integrity of MA data, secrecy of path of the MA, and nonrepudiation of data are all addressed in [31]. Trusted platforms verify the integrity of data, detect any alteration, insertion, and deletion to the collected data, and inform the owner platform of any changes. After the execution of the MA, the platform sends its output data to the trusted platform and then migrates to the next destination. When the agent returns to its owner carrying the collected data, a hash of the data collected at each platform is sent to the trusted platform. The trusted platform verifies the integrity of collected data and informs the owner of any malicious changes detected. The protocol is based on multihop architecture where an MA migrates from one platform to another to collect data. However, the loss of the MA results in the final loss of all the collected data.

The protocol in [32] relies on a Knowledge-Based System (KBS) that collects information about hosts' trustworthiness. The protocol is based on trusted hosts and chain relations. The MA starts at the owner host and with the help of the KBS, it chooses its next destination. The MA travel to hosts and collects data. The visited trust hosts check the integrity of all the data collected by the MA. If any modification is detected, the owner host is informed. Asymmetric encryptions and hash functions are used to provide data confidentiality, nonrepudiation, and integrity. Moreover, owner's identity and path taken by the MA are encrypted and can only be accessed by the owner and trust hosts. However, the system's complexity is high and introduces an operational overhead because of the expensive integrity check at trust hosts. Moreover, again the loss of the agent results in the loss of all the collected data and nonfunctionality of the system [44].

The protocol proposed in [33] splits the agent code into two codes, critical and normal. The critical code is only executed at the trusted platform and the normal code is carried by the MA. The system has two trusted platforms: one is responsible for executing the critical code, while the other watches and provides the system with a recovery policy in case of a breakdown. The MA collects the data and sends it to the trusted platform to get the final result. The trusted platform starts a timer to detect replay attacks. The protocol provides confidentiality of data, integrity of data and code, and the possibility of recovering the agent code in case of modification or breakdown.

A symmetric cryptography scheme is proposed in [34]. The generation and distribution of the symmetric keys are unique. Keys are derived from data generated at platforms at run time. Therefore, it becomes harder for attackers to guess or predict the keys. Agent owner authentication, confidentiality, and integrity of both code and data are provided. However, the protocol is vulnerable to replay attacks and the system functionality is affected by the loss of MA and its data.

Despite the usefulness of the aforementioned protocols, they cannot satisfy all the security requirements. Moreover, as we have seen, they are impaired by some hidden assumptions and drawbacks. In order to overcome issues of the current state of the art, we propose BROSMAP which provides improvements over the previously reviewed protocols. It provides mutual authentication between users and service providers, integrity, confidentiality, accountability, authorization, and nonrepudiation. Another important feature that distinguishes BROSMAP from other protocols is its ability to continue its functionality even if one of the agents is killed. BROSMAP satisfies the security requirements and protection from MITM, replay, masquerade, unauthorized access, and modification attacks. A detailed comparison of BROSMAP with the reviewed protocols will be presented later in Section 5. In the next section, a detailed description of BROSMAP is discussed, followed by security analysis and results of carried formal verification test, the performance evaluation, and implementation details with JADE.

4. Proposed BROSMAP

BROSMAP provides fundamental security requirements and protection from different attacks. The proposed broadcast architecture allows the system to stay functioning even if one of the MAs is maliciously killed or lost and maintain good performance at the same time.

In BROSMAP, the hybrid approach of using both symmetrical and asymmetrical cryptography is used. Using the asymmetrical cryptography enables the system to provide many security requirements that are harder to cover with symmetrical cryptography such as mutual authentication, accountability, and nonrepudiation. On the other hand, using symmetrical cryptography enhances the system performance

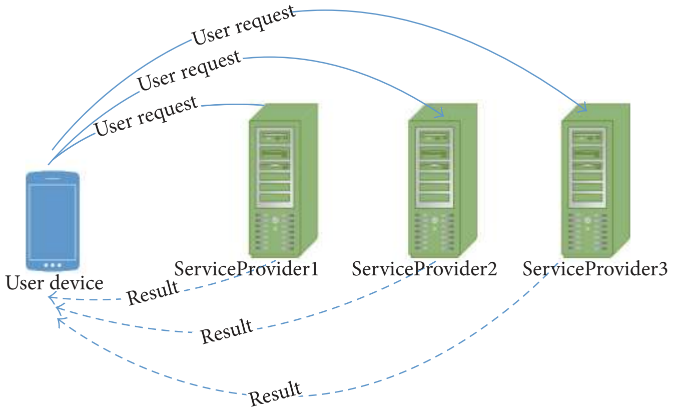


Figure 1: System scenario of BROSMAP in a distributed application.

as it is lighter operational wise. Therefore, incorporating the hybrid approach provides the system with the needed security requirements maintaining the good performance at the same time.

In order to explain how the interaction happens in further detail, notations and definitions are provided in (I) in "Notations" section.

An illustrative scenario of the proposed BROSMAP for distributed applications is shown in Figure 1. A scenario of a LBS application is used as a possible example of a distrusted application. Service providers are registered after being verified by the telecomm operator. Service providers provide their identity, address, and type of services they offer. Their reputation can also be an important factor in the registration process. We also assume that PKI has been set up. In our scenario, service providers provide users with names of possible hotels, restaurants, and flight agencies that are close to the specified location. Users are provided with different available options along with additional information about the option such as average price and distance from current location. The provided information helps users in the decision making process.

Accessing the application, the user provides his/her request. Then, the request is broadcast to registered service providers. Each service provider receives the request and searches its database for matching results to be sent back. If one of the broadcast messages is lost, the user is still able to get results from the other service providers. An example of a user request would be "provide me with names of close by hotels" and a possible answer would be "Sheraton Hotel is 10 miles away and cost on average $100 per night."

BROSMAP system architecture is shown in Figure 2. PLA_1 is the owner platform that launches copies of the MA carrying the user request containing the location and service type to service providers. As shown, the system includes $i-1$ service providers platforms which are PLA_2 to PLA_i. The owner platform PLA_1 generates MA_1. MA_1 carries the user request R to collect output of requests OR_i from platforms, containing possible service options for users.

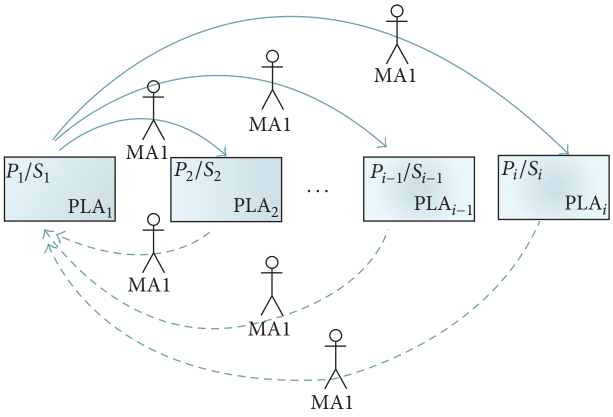


Figure 2: BROSMAP system architecture.

Next, steps of interaction between owner platform and service providers are explained in detail.

4.1. Stages of Interaction between Owner and Service Provider Platforms. For the sake of simplicity, here, we do not discuss certificates distribution as we assume that platforms have been verified and granted certificates by the certificate authority (CA) [45–48]. The owner platform has a prior knowledge of the available trusted service providers to be contacted for results. Service providers are service-specific servers providing information to the user. For example, some servers hold information about restaurants, some others about hotels, and so on; therefore, service providers are chosen according to the requested service. Moreover, BROSMAP sheds light on securing the MA during migration and interaction with platforms; however, the exchange and verification of certificates are out of the scope of this paper.

Next, stages of interaction between owner platform PLA_1 and PLA_n are described. PLA_n is any service provider in the system where i and n are positive integers and $2 \leq n \leq i$:

(1) Starting stage at the owner platform. The system interaction starts when the user provides the application with a request. The owner platform is the starting point. The first stage of the interaction is the preparation and broadcast of user request message to service providers platforms steps are summarized as follows:

   (i) the owner platform PLA_1 does the following:

      (a) it gets request R from user;
      (b) it generates time stamp T_1;
      (c) it generates session key K_n;
      (d) it generates $h(R \parallel T_1)$;
      (e) it encrypts $h(R \parallel T_1)$ with K_n to generate $K_n(h(R \parallel T_1))$;
      (f) it signs $K_n(h(R \parallel T_1))$ and add_n with S_1 to generate $S_1(K_n(h(R \parallel T_1)) \parallel \text{add}_n)$;
      (g) it encrypts R, add_n, K_1 and T_1 with P_n to generate $P_n(R \parallel \text{add}_n \parallel K_n \parallel T_1)$.

(2) Sending request to service provider platform.

   (i) $\text{PLA}_1 \rightarrow \text{PLA}_n$;
   $P_n(R \parallel \text{add}_n \parallel K_n \parallel T_1) \parallel S_1(K_n(h(R \parallel T_1)) \parallel \text{add}_n)$.

(3) Processing of user request at the service provider platform. The second stage involves processing the received user requests messages at the service provider platforms:

   (i) the service provider platform PLA_n does the following:

      (a) it decrypts $P_n(R \parallel \text{add}_n \parallel K_n \parallel T_1)$ with S_n to get the session key K_n;
      (b) it checks freshness of T_1;
      (c) it decrypts $S_1(K_n(h(R \parallel T_1)) \parallel \text{add}_n)$ with P_1 to get $(K_n(h(R \parallel T_1)) \parallel \text{add}_n)$;
      (d) it decrypts with K_n to get $h(R \parallel T_1)$;
      (e) it checks if $h(T_1 \parallel R)$ in $K_n(h(R \parallel T_1))$ matches hash of sent T_1 and R in $P_n(R \parallel \text{add}_n \parallel K_n \parallel T_1)$;
      (f) if all checks are correct, then it executes agent and generates result OR_n;
      (g) it generates time stamp T_n;
      (h) it generates hash of the result $h(\text{OR}_n)$;
      (i) it generates hash of the time stamp $h(T_n)$;
      (j) it encrypts result OR_n, R, add_1, T_1, and T_n with the session key K_n to get $K_n(\text{OR}_n \parallel R \parallel \text{add}_1 \parallel T_1 \parallel T_n)$;
      (k) it signs $(h(R \parallel T_1) \parallel h(\text{OR}_n) \parallel h(T_n))$ with S_n to get $S_n(h(R \parallel T_1) \parallel h(\text{OR}_n) \parallel h(T_n))$.

(4) Sending results to owner platform.

   (i) $\text{PLA}_n \rightarrow \text{PLA}_1$;
   $K_n(\text{OR}_n \parallel R \parallel \text{add}_1 \parallel T_1 \parallel T_n) \parallel S_n(h(R \parallel T_1) \parallel h(\text{OR}_n) \parallel h(T_n))$.

(5) Processing results at owner platform. Finally, in the last stage, the owner platform receives and processes the reply messages carrying results:

   (i) platform PLA_1 does the following:
      (a) it decrypts $K_n(\text{OR}_n \parallel R \parallel \text{add}_1 \parallel T_1 \parallel T_n)$ with K_n;
      (b) it checks freshness of T_1 and T_n;
      (c) it decrypts $S_n(h(R \parallel T_1) \parallel h(\text{OR}_n) \parallel h(T_n))$ with P_n;
      (d) it checks if $h(R \parallel T_1)$ in $S_n(h(R \parallel T_1) \parallel h(T_n) \parallel (h(\text{OR}_n)))$ matches $h(R \parallel T_1)$ in $K_n(\text{OR}_n \parallel R \parallel \text{add}_1 \parallel T_1 \parallel T_n)$;
      (e) it checks if $h(T_n)$ in $S_n(h(R \parallel T_1) \parallel h(T_n) \parallel (h(\text{OR}_n)))$ matches $h(T_n)$ in $K_n(\text{OR}_n \parallel R \parallel \text{add}_1 \parallel T_1 \parallel T_n)$;
      (f) it checks if $h(\text{OR}_n)$ in $S_n(h(R \parallel T_1) \parallel h(T_n) \parallel (h(\text{OR}_n)))$ matches $h(\text{OR}_n)$ in $K_n(\text{OR}_n \parallel R \parallel \text{add}_1 \parallel T_1 \parallel T_n)$;
      (g) if all checks are correct, it displays results to the user.

Figure 3 shows the messages exchanged between owner platform PLA_1 and each service provider PLA_n in the system.

5. Security Analysis of BROSMAP

In this section, a security analysis of BROSMAP is provided. We show that BROSMAP provides the following security requirements:

(i) Mutual authentication: mutual authentication is the assurance that communicating entities can verify each other's identity. Take, for example, the interaction between PLA_1 and PLA_n. Both platforms are able to identify each other through the PKI using the signatures and certificates. Also they are sure of who sent the message because of the digital signature. The messages sent are fresh due to the use of time stamps and both platforms can verify that the messages exchanged are actually intended for them because of the concatenation of the address of the destination in the message. Moreover, messages are received in order, and PLA_n replies to PLA_1 after it receives the request. The interaction provides all the requirements for mutual authentication; therefore, BROSMAP provides mutual authentication.

(ii) Confidentiality: the exchanged information between platforms remains secret throughout the communication due to the use of public and symmetric key encryptions. At any time, only the owner platform can access the output of the request because of the encryption with the secret session key. In addition, the user request and time stamps are also encrypted and remain confidential. The required information for decryption requires the knowledge of either the private key or the session key.

(iii) Integrity: in BROSMAP, the use of one-way hash function and signature supports a standard integrity

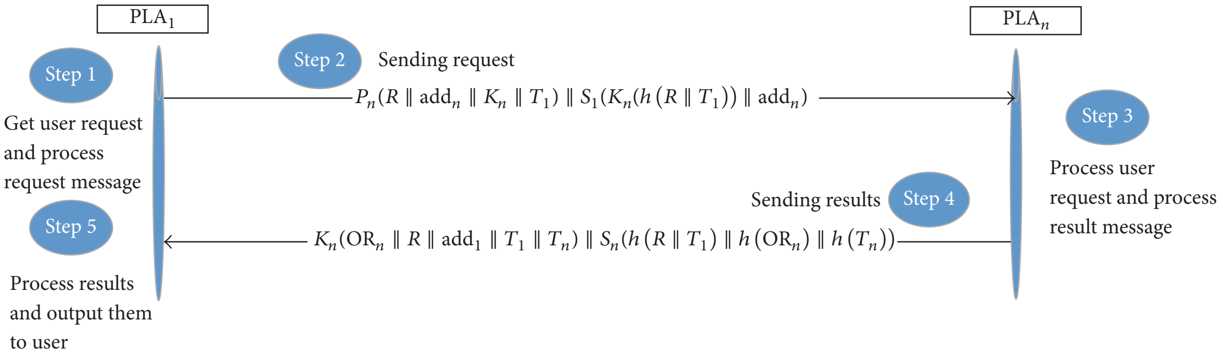

PLA$_1$
PLA$_n$
Step 1
Get user request and process request message
Step 2
Sending request
$P_n(R \parallel \text{add}_n \parallel K_n \parallel T_1) \parallel S_1(K_n(h(R \parallel T_1)) \parallel \text{add}_n)$
Step 3
Process user request and process result message
Sending results
Step 4
$K_n(\text{OR}_n \parallel R \parallel \text{add}_1 \parallel T_1 \parallel T_n) \parallel S_n(h(R \parallel T_1) \parallel h(\text{OR}_n) \parallel h(T_n))$
Step 5
Process results and output them to user

Figure 3: Summary of messages exchanged between PLA$_1$ and PLA$_n$ through MA.

test. The integrity of request, the output of the request, and all the time stamps are guaranteed at all times. Any change or manipulation is detected.

(iv) Accountability: BROSMAP provides accountability in the sense that if platform PLA$_2$ decides to act maliciously and change the request, then PLA$_1$ detects the change comparing the hashes, and PLA$_2$ signature provides proof of its malicious action. Our protocol is proposed for distributed services applications; therefore, the architecture is made up of both users (senders) and service providers (servers). The request is broadcast from the user to the service providers. Accountability is provided to hold service providers accountable if they acted maliciously and decided, for example, to send false data or change the user request. At this stage, the user needs to know who did the malicious action and might decide to stop dealing with that server in future requests.

(v) Authorization: when the user accesses the application and provides his/her credentials, this information is verified to decide whether to grant the user access to the application services or not. In addition, the certificate authority checks the validity of the certificate and the identity of the platforms; therefore, only authorized platforms (service providers) provide the user with the service. Moreover, by providing mutual authentication, platforms are able to verify each other and any communication request or message sent by a nontrusted platform is ignored. The request, time stamps, and the output of the request are all confidential and can only be decrypted by authorized platforms.

(vi) Nonrepudiation: the signature of platforms provides proof of the originator of the message sent. In BROSMAP, platforms sign messages with their private keys providing the possibility for action traceability.

The protocol also provides protection from different attacks:

(i) Replay attacks: the time stamp indicates the freshness of the message and any expired message is discarded. Both the freshness of the data and delivery are checked. Two timestamps are used. The first one is generated to stamp the time of the request generation and the second one is generated to stamp the time of the data (results) generation. The user will only consider the delivered results if and only if both timestamps are fresh; that is, the request and generated data are fresh and new.

(ii) MITM attacks: in BROSMAP, even if a malicious platform gets hold of exchanged messages, it cannot decrypt and read their content. Moreover, all data is encrypted and remains confidential. Integrity checks are provided to detect any modification. Therefore, MITM attacks are detected and fake messages or requests are dropped.

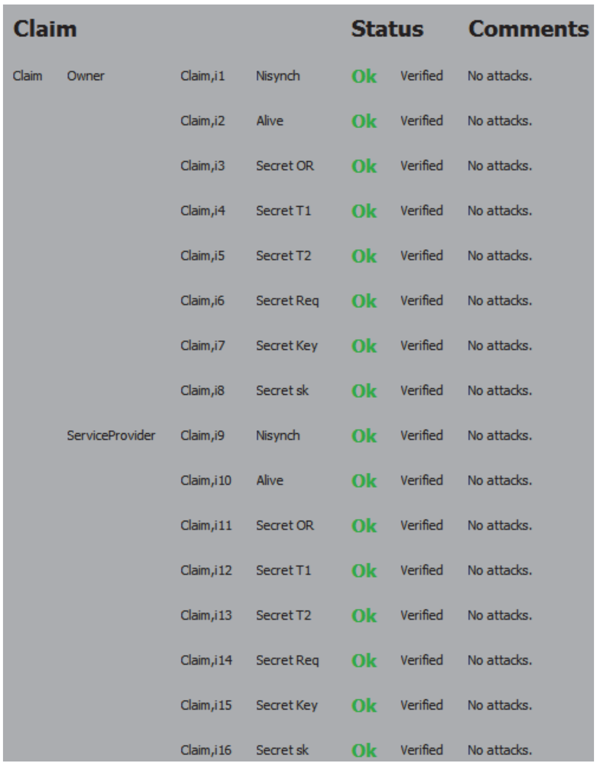
(iii) Modification attacks: a malicious platform or agent cannot modify the time stamps, the request, and the results without being detected, due to the digital signature and the integrity checks comparing hash values at each platform.

(iv) Masquerade attacks: a malicious platform cannot impersonate another platform because of the use of the digital signatures.

To verify the correctness of BROSMAP, formal methods verification proof is carried using Scyther. Next, a detailed explanation of the verification tests and results are presented along with a comparison of BROSMAP and other existing protocols.

5.1. Formal Verification of BROSMAP

To verify BROSMAP, we selected Scyther formal methods verification tool. Scyther is a free formal analysis tool used to verify security protocols. Scyther provides a set of claims to test secrecy of information, synchronization, and authentication between communication parties. Synchronization means that messages sent are received by the intended communication party and that received messages are actually sent by the communication party. It also means that the messages sent and received are in order and their content has not been modified. Scyther also tests for aliveness of a communication party, which basically means that the party is alive and has executed some events [49–52].



Claim				Status		Comments
Claim	Owner	Claim,i1	Nisynch	Ok	Verified	No attacks.
		Claim,i2	Alive	Ok	Verified	No attacks.
		Claim,i3	Secret OR	Ok	Verified	No attacks.
		Claim,i4	Secret T1	Ok	Verified	No attacks.
		Claim,i5	Secret T2	Ok	Verified	No attacks.
		Claim,i6	Secret Req	Ok	Verified	No attacks.
		Claim,i7	Secret Key	Ok	Verified	No attacks.
		Claim,i8	Secret sk	Ok	Verified	No attacks.
	ServiceProvider	Claim,i9	Nisynch	Ok	Verified	No attacks.
		Claim,i10	Alive	Ok	Verified	No attacks.
		Claim,i11	Secret OR	Ok	Verified	No attacks.
		Claim,i12	Secret T1	Ok	Verified	No attacks.
		Claim,i13	Secret T2	Ok	Verified	No attacks.
		Claim,i14	Secret Req	Ok	Verified	No attacks.
		Claim,i15	Secret Key	Ok	Verified	No attacks.
		Claim,i16	Secret sk	Ok	Verified	No attacks.

Figure 4: Output of Scyther claims test.

5.1.1. Verification Code. To verify BROSMAP, we set up a set of claims testing the secrecy of information and synchronization in the code. A set of claims tests the secrecy of the session key, private keys, request, output of request, and time stamps at the owner and the service provider platforms. Moreover, other claims test synchronization between the owner and the service provider platforms.

The verification code, attached in Algorithm1, represents the protocol description in Security Protocol Description Language (SPDL) [50]. The code has two roles, the owner platform role and the service provider role. The code shows the exchanged messages in addition to the set of claims that need to be verified.

Figure 4 shows the output of the verification test for the defined claims. The test verifies the secrecy of the exchanged time stamps, request, output of request, private keys, and session key. Moreover, aliveness of parties and synchronization between them are also verified.

Scyther generates trace figures (Figures 5–7) showing the interaction and exchanged messages between communication parties. BROSMAP interaction can be summarized into three stages. Each stage is shown through a trace figure. Because Scyther uses its own notations that are different than the ones presented earlier, (II) in "Notations and Definitions" section reports the new notations used in Scyther traces.

Figure 5 explains the first stage, which is the broadcast of the user request to the service provider platform. As shown in the figure, owner platform (Owner#1) sends the message to the service provider platform (ServiceProvider#1).

The second stage is shown in Figure 6. In this stage, ServiceProvider#1 processes the request message sent by Owner#1. ServiceProvider#1 authenticates Owner#1 and checks the integrity of sent information through the match functions. Match functions operate as a logical comparison that compares two values to check if they are equal. Match functions are used to check the integrity of the data and time stamps through comparing the hash sent in the message with the hash of the newly received data, generated on the current platform. After passing the integrity test, ServiceProvider#1 generates and sends the results back to Owner#1.

The final stage is shown in Figure 7, where Owner#1 receives the message sent by ServiceProvider#1 and checks the integrity of its content through the match functions. The three generated traces show that message synchronization exists between the interacting parties. Moreover, it also proves the aliveness and reachability of parties.

5.1.2. Analysis of Results. In this section we discuss the mapping between the Scyther test results and the fundamental security requirements:

(i) Mutual authentication: synchronization between the owner and the service provider platforms is verified through claim_{i1}, claim_{i2}, claim_{i9}, and claim_{i10} shown in Algorithm 1 in lines 21, 22, 42, and 43, respectively. Synchronization ensures that messages sent and received by intended parties are unmodified and in order. Also, the use of PKI and ensuring the secrecy of private keys provides the requirements to identify platforms and, therefore, mutual authentication exists between the two parties.

(ii) Confidentiality: time stamps, request, output of request, private key, and session key exchanged are verified for secrecy through claim_{i3} to claim_{i8} and claim_{i11} to claim_{i16} shown in lines 23–28 and 44–49, respectively. The data remains confidential throughout the communication.

(iii) Integrity: hash function and match operations are used to verify the integrity of the request, time stamps, and output of request. The match functions shown in lines 18–20 and 39 detect any change in the data.

(iv) Authorization: the exchanged information remains secret throughout the interaction and only authorized parties with proper decryption information can access it. This is proven through ensuring the secrecy of the exchanged session key through claim_{i7} in line 27 and claim_{i15} in line 48. Moreover, mutual authentication guarantees that the interaction happens between trusted parties, hence providing protection from unauthorized accesses.

(v) Accountability and nonrepudiation: using signature and verifying the secrecy of parties' private keys in claim_{i8} in line 28 and claim_{i16} in line 49, a party can be accounted for its malicious actions such as making changes to time stamp or request. Moreover, nonrepudiation is also provided in the sense that no party can deny data sent by it.

```
usertype TimeStamp,Data,Request,Key;
hashfunction H;
macro hash1=H(Req,T1);                          % hash the request and time stamp 1
macro hashofdata=H(OR);                         % hash the result
macro hashoftime=H(T2);                         % hash the request and time stamp 2
protocol MyProtocol(Owner, ServiceProvider){
role Owner { fresh kir:Key;                     % generate session key
fresh T1:TimeStamp;                             % generate time stamp 1
var T2:TimeStamp;                               % prepare variable to receive time stamp 2
const Req:Request;                              % generate request
var OR:Data; % prepare variable to receive result
send_1(Owner, ServiceProvider,
{{hash1}kir, ServiceProvider}sk(Owner),{Req, ServiceProvider, kir, T1}pk(ServiceProvider));
recv_10(ServiceProvider, Owner, {hash1, hashofdata, hashoftime}sk(ServiceProvider),
{OR, Req, Owner, T1, T2}kir);
macrohashofdata2 = H(OR);                       % generate hash of received result
macrohashoftime2 = H(T2);                       % generate hash of received timestamp
match(hash1, hash2);                            % compare hashes
match(hashofdata, hashofdata2);
match(hashoftime, hashoftime2);
claim_i1(Owner, Nisynch); % test Claims
claim_i2(Owner, Alive);
claim_i3(Owner, Secret, OR);
claim_i4(Owner, Secret, T1);
claim_i5(Owner, Secret, T2);
claim_i6(Owner, Secret, Req);
claim_i7(Owner, Secret, Key);
claim_i8(Owner, Secret, sk);
}
role ServiceProvider{
fresh OR:Data;                                  % generate result
var kir:Key;                                    % prepare variable to receive session key
fresh T2:TimeStamp;                             % generate time stamp 2
var T1:TimeStamp;                               % prepare variable to receive time stamp 1
var Req:Request;                                % prepare variable to receive request
recv_1(Owner, ServiceProvider,
{{hash1}kir, ServiceProvider}sk(Owner), {Req, ServiceProvider, kir, T1}pk(ServiceProvider);
macro h2=H(Req,T1);                             % generate hash of received request and time stamp 1
match (hash1,h2);                               % compare hashes
send_10(ServiceProvider, Owner,
{hash1, hashofdata, hashoftime}sk(ServiceProvider), {OR, Req, Owner, T1, T2}kir);
claim_i9(ServiceProvider, Nisynch);             % test claims
claim_i10(ServiceProvider, Alive);
claim_i11(ServiceProvider, Secret, OR);
claim_i12(ServiceProvider, Secret, T1);
claim_i13(ServiceProvider, Secret, T2);
claim_i14(ServiceProvider, Secret, Req);
claim_i15(ServiceProvider, Secret, Key);
claim_i16(ServiceProvider, Secret, sk);
}}
```

Algorithm 1

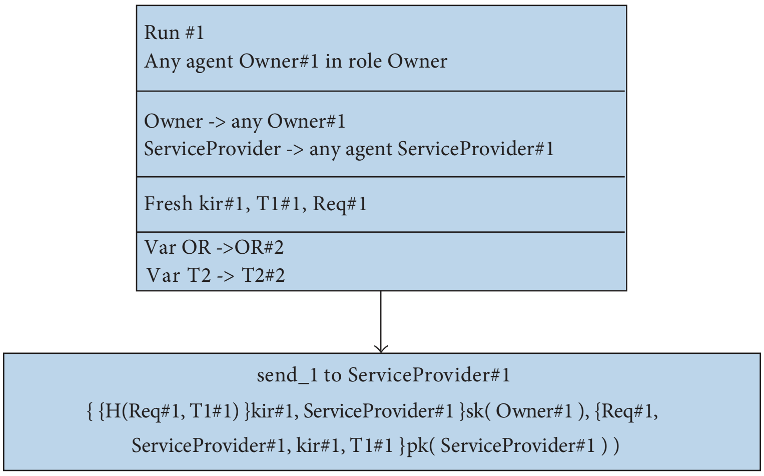

Run #1
Any agent Owner#1 in role Owner
Owner -> any Owner#1
ServiceProvider -> any agent ServiceProvider#1
Fresh kir#1, T1#1, Req#1
Var OR ->OR#2
Var T2 -> T2#2
send_1 to ServiceProvider#1
{ {H(Req#1, T1#1) }kir#1, ServiceProvider#1 }sk(Owner#1), {Req#1, ServiceProvider#1, kir#1, T1#1 }pk(ServiceProvider#1))

Figure 5: Stage 1: Owner platform: Owner#1 sends request to ServiceProvider#1.

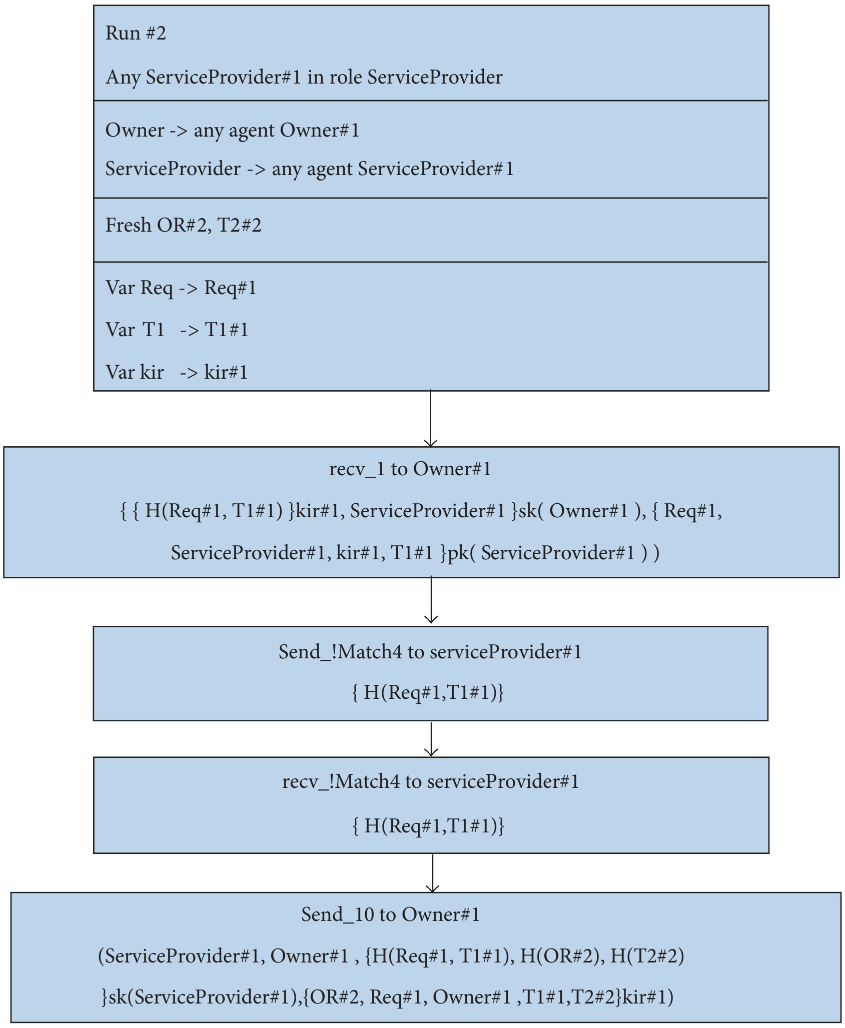

Run #2
Any ServiceProvider#1 in role ServiceProvider
Owner -> any agent Owner#1
ServiceProvider -> any agent ServiceProvider#1
Fresh OR#2, T2#2
Var Req -> Req#1
Var T1 -> T1#1
Var kir -> kir#1
recv_1 to Owner#1
{ { H(Req#1, T1#1) }kir#1, ServiceProvider#1 }sk(Owner#1), { Req#1, ServiceProvider#1, kir#1, T1#1 }pk(ServiceProvider#1))
Send_!Match4 to serviceProvider#1
{ H(Req#1,T1#1)}
recv_!Match4 to serviceProvider#1
{ H(Req#1,T1#1)}
Send_10 to Owner#1
(ServiceProvider#1, Owner#1 , {H(Req#1, T1#1), H(OR#2), H(T2#2) }sk(ServiceProvider#1),{OR#2, Req#1, Owner#1 ,T1#1,T2#2}kir#1)

Figure 6: Stage 2: ServiceProvider#1 processes the user request and sends result back to Owner#1.

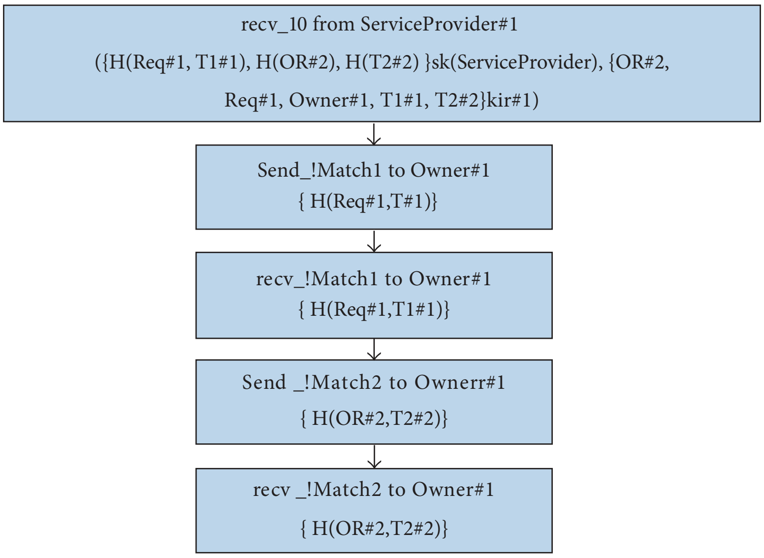

recv_10 from ServiceProvider#1
({H(Req#1, T1#1), H(OR#2), H(T2#2) }sk(ServiceProvider), {OR#2, Req#1, Owner#1, T1#1, T2#2}kir#1)
Send_!Match1 to Owner#1
{ H(Req#1,T#1)}
recv_!Match1 to Owner#1
{ H(Req#1,T1#1)}
Send _!Match2 to Ownerr#1
{ H(OR#2,T2#2)}
recv _!Match2 to Owner#1
{ H(OR#2,T2#2)}

Figure 7: Stage 3: Owner#1 processes the output of the result coming from ServiceProvider#1.

Figure 4 shows that according to Scyther, BROSMAP satisfies all the desired requirements. Also, we can state that BROSMAP provides protection from different types of attacks. Using time stamps and ensuring their secrecy and integrity provide protection from replay attacks. Synchronization and verification of secrecy of information ensure that communication between parties is secure and that there is no possibility for MITM attacks to pass without being detected. Moreover, the use of signature provides protection from masquerade attacks. A malicious party cannot masquerade as another legitimate party unless it gets hold of its private key; however, it is verified through claim$_{i8}$ and claim$_{i16}$ that private keys remain confidential throughout the communication. In addition, any malicious unauthorized modification of time stamps, results, and requests is detected and the message is discarded; therefore, modification attacks are also eliminated.

A comparison between protocols in related work and BROSMAP is carried out. The criteria scheme used for comparison is whether the protocol provides the following security requirements: anonymity, authentication, authorization, accountability, confidentiality, integrity, and nonrepudiation. None of the proposed works provides availability; therefore it is not included in Table 1.

Table 1 shows the detailed comparison. As shown, most of the compared works passed two or three of the seven security requirements, while [31] is found to provide five. BROSMAP, however, is the only protocol that provides six of the seven requirements. BROSMAP provides mutual authentication, authorization, accountability, confidentiality, integrity, and nonrepudiation.

Another two important comparison factors are whether the protocol takes into consideration the effect of potential loss or killing of an agent and if the proposed protocol is verified by any security verification tool. These two factors are important because the first affects the system functionality and the other verifies the correctness of the proposed protocol. In the related works, only [31, 33] consider agent loss by proposing the matrix hop and recovery method, respectively. Moreover, the only work that verified its proposed protocol is [28] and was proven to withstand all simulated replay attacks in JADE; however, none of the other works used formal verification methods. In BROSMAP, the two criteria are also covered. BROSMAP considers agent loss in the system by broadcasting simultaneously. Moreover, its correctness is proven through formal verification proof with Scyther as was previously explained in this section. Therefore, among the compared works BROSMAP is the only protocol that provides a proof of correctness through formal verification method.

In the next section, the performance of BROSMAP is evaluated and compared to the other protocols.

6. Performance Evaluation of BROSMAP

To evaluate the performance of BROSMAP, the total number of operations is approximated and compared to the other protocols. To give a fair comparison between BROSMAP and the other protocols, the widely accepted evaluation method used in [23] is applied. According to [23, 53], the computational cost of an asymmetric operation (A) is equivalent to one point operation which is equivalent to 1000 symmetrical operations (S) and 10000 hash operations (H). Therefore, every asymmetrical, symmetrical, and hash operation is evaluated as 1, 0.001, and 0.0001 point operations, respectively.

Besides BROSMAP, the total number of operations is approximated for the following schemes [27, 31–34]. These schemes are chosen because they are similar to BROSMAP in the sense that they are also based on cryptographic techniques. The total number of operations depends on N, which is the number of platforms the MA visits to collect results in a MAS. In Table 2, the total number of operations for each scheme is calculated.

Table 1: Comparison of related works and BROSMAP.

Related works	Anonymity	Authentication	Authorization	Accountability	Confidentiality	Integrity	Nonrepudiation
Raji and Tork Ladani, 2010 [22]	Yes	No	Yes	No	Yes	No	No
Kuo et al., 2014 [23]	Yes	Yes (mutual)	Yes	No	No	No	No
Leszczyna, 2007 [24]	Yes	No	No	No	No	No	No
Garrigues et al., 2008 [25]	No	Yes (mutual)	Yes	No	No	No	No
Garrigues et al., 2009 [28]	No	Yes (mutual)	Yes	No	No	No	No
Fong et al., 2011 [26]	No	(one-way)	Yes	No	Yes	No	No
Venkatesan et al., 2010 (MIP) [27]	No	(one-way)	Yes	No	No	No	No
Venkatesan et al., 2010 (XRC) [27]	No	(one-way)	No	No	No	No	Yes
Kapnoullas et al., 2003 [41]	No	(one-way)	Yes	No	No	No	Yes
Dhanalakshmi and Nawaz, 2012 [29]	No	No	No	No	No	No	No
Sulaiman et al., 2009 [43]	No	Yes (mutual)	Yes	No	Yes	No	No
Guan et al., 2007 [31]	No	Yes (mutual)	Yes	No	Yes	Yes	Yes
Geetha and Jayakumar, 2011 [32]	Yes	No	No	No	Yes	Yes	Yes
Ouardani et al., 2007 [33]	No	Yes (mutual)	No	No	Yes	Yes	No
Srivastava and Nandi, 2014 [34]	No	(one-way)	No	No	Yes	Yes	No
BROSMAP	No	Yes (mutual)	Yes	Yes	Yes	Yes	Yes

Table 2: Comparison of related protocols according to number of operations.

Scheme	Total operations for system with N platforms
Guan et al., 2007 [31]	$(11A + 4H) \times N + 6A + 2H = 11.0004 \times N + 6.002$
Srivastava and Nandi, 2014 [34]	$(6A + 2S + 6H) \times N + 4A + S + 2H = 6.0205 \times N + 4.0012$
Ouardani et al., 2007 [33]	$12A \times N = 12 \times N$
Venkatesan et al., 2010 (XRC) [27]	$(3A + H) \times N + (1A + H) = 3.0001 \times N + 1.0001$
Geetha and Jayakumar, 2011 [32]	$(8A + 3H) \times N + (2A + H) = 8.0003 \times N + 2.0001$
BROSMAP	$(6A + 4S + 6H) \times N = 6.0046 \times N$

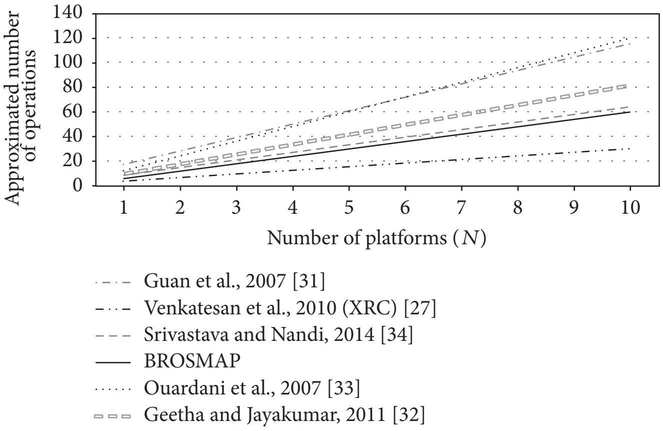


Figure 8: Performance of BROSMAP compared to other proposed protocols.

To evaluate the performance of protocols, the increase in the number of operations as the number of service providers N increases is studied. Figure 8 shows the change in the total number of approximated operations as N increases.

The key point that affects performance is the number of asymmetrical operations as they have the highest computational cost. The protocol in [33] is based on asymmetrical operations only and protocol in [31] is based on asymmetrical and hash operations. For this reason, they show the greatest increase in the number of operations as N increases. The protocol in [27] is also an asymmetrical based scheme; however protocol in [27] uses digital signature to provide integrity to agent code and protection from nonrepudiation attacks only. The approach in [32] uses asymmetrical and hash operations to provide anonymity, confidentiality, integrity, and nonrepudiation. Moreover, the protocol performance not only is dependent on N but also depends on a number of trusted hosts that the MA visits during its journey. To approximate the number of operations for this scheme, we assume that only one trust host is visited; therefore, the approximation represents the minimum number of operations the system can have. The other schemes rely on the hybrid approach which, as Figure 8 shows, is more efficient; it has a lower overall computational cost compared to the other schemes. Despite the fact that the approach in [34] is multihop and BROSMAP is based on broadcast architecture, BROSMAP performs better than [34] as the number of platforms increases. Moreover, as previously mentioned, the protocol provides one-way authentication, confidentiality, and integrity only. It is also vulnerable to replay attacks and does not consider the case of loss or malicious killing of MA causing the system to lose all the data collected by the MA. BROSMAP provides mutual authentication, authorization, integrity, confidentiality, accountability, and nonrepudiation. In addition, it also provides protection from different attacks such as replay, MITM, masquerade, and repudiation attacks. In BROSMAP, malicious killing of an MA in the system does not mean the loss of the whole data, but rather part of the data is lost; however, there are other copies of MAs that can compensate and carry on the system functionality. Taking all that into consideration, the number of operations of BROSMAP is acceptable. While BROSMAP is not the most efficient protocol, the performance of the system seems reasonable compared to other schemes that satisfy fewer security requirements.

7. Implementation of Possible Distributed Service Application: LBS

To verify the viability of BROSMAP, implementation of an example of a LBS application that uses BROSMAP for securing its communication on the well-known JADE [16, 17] platform is provided. In this paper, LBS applications were chosen because of their popularity among users; however, it is important to note that the proposed BROSMAP can be used with any other distributed application. According to the scenario of Section 4, the application provides users with

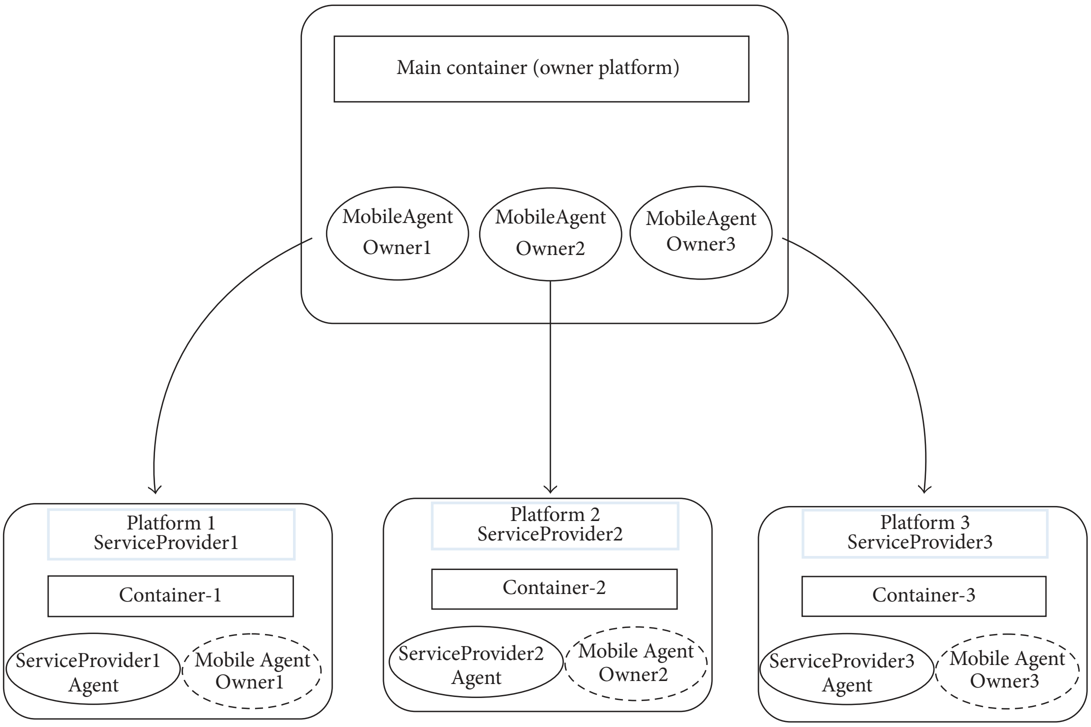


Figure 9: Basic architecture of JADE setup.

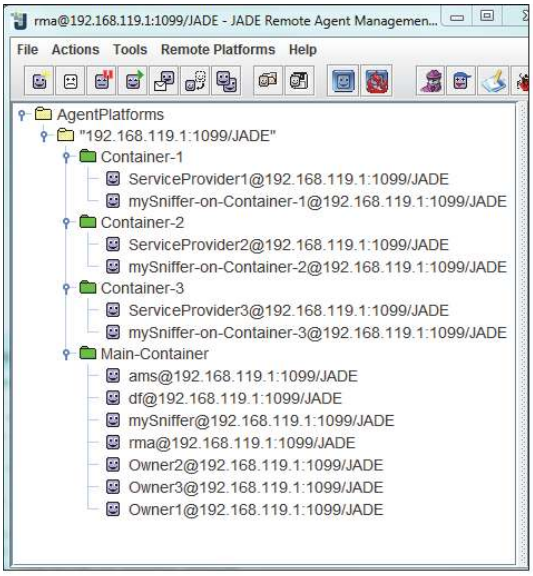
names of possible hotels, restaurants, and flight agencies that are close to the specified location. Users are provided with different options and some information about each option, such as the average price and distance from current location. To simulate the application, four containers were created the MainContainer, Container-1, Container-2, and Container-3; see Figure 9.

MainContainer represents the owner platform, while the other containers are the service providers platforms. Both mobile and stationary agents are used. MAs called owner agents carry the owner request to the service providers to collect results. Other agents reside in the service provider platform that communicates with owner MAs to process the request and exchange results. At the MainContainer, three MAs of the type owner are created, Owner1, Owner2, and Owner3.

The user request is broadcast where each copy of owner agent carries the user request to a different service provider. Three stationary agents (ServiceProvider1, ServiceProvider2, and ServiceProvider3) are created at each service provider. Figure 10 shows the containers and agents in JADE prior to the interaction process.

JADE provides the user with the option of sniffing messages exchanged between two parties. A sniffer agent, mySniffer, is created at each container to trace interactions between the owner agents and service providers' agents. Figure 11 shows sniffed messages that are exchanged between the owner and service providers during the process of the user request and sending of results.

The content of one of the sniffed messages from Owner1 to ServiceProvider1 is shown in Figure 12. Message content is encrypted and can only be read after proper decryption at the intended destination.


rma@192.168.119.1:1099/JADE - JADE Remote Agent Managemen...
File
Actions
Tools
Remote Platforms
Help

```
AgentPlatforms
  "192.168.119.1:1099/JADE"
    Container-1
      ServiceProvider1@192.168.119.1:1099/JADE
      mySniffer-on-Container-1@192.168.119.1:1099/JADE
    Container-2
      ServiceProvider2@192.168.119.1:1099/JADE
      mySniffer-on-Container-2@192.168.119.1:1099/JADE
    Container-3
      ServiceProvider3@192.168.119.1:1099/JADE
      mySniffer-on-Container-3@192.168.119.1:1099/JADE
    Main-Container
      ams@192.168.119.1:1099/JADE
      df@192.168.119.1:1099/JADE
      mySniffer@192.168.119.1:1099/JADE
      rma@192.168.119.1:1099/JADE
      Owner2@192.168.119.1:1099/JADE
      Owner3@192.168.119.1:1099/JADE
      Owner1@192.168.119.1:1099/JADE
```

Figure 10: Simulated container and agents in JADE.

Figure 13 shows the result of a successful interaction between the owner and service providers. The final results collected by MAs are printed to the user containing the name of the hotel, distance from location, and approximate average price for one night stay.

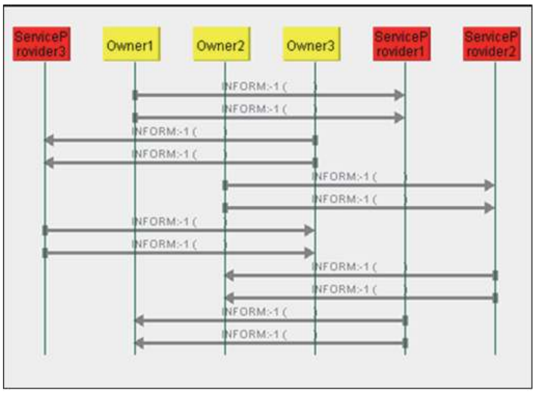
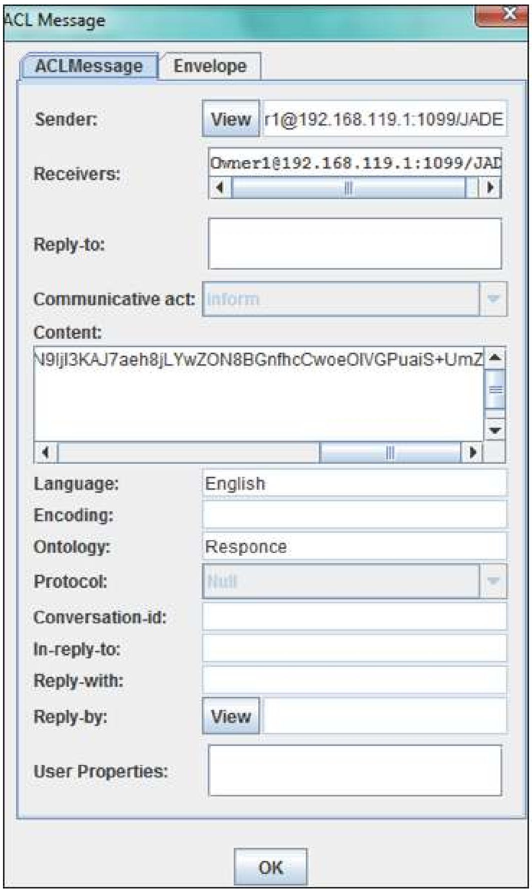


Figure 11: Traced exchanged messages in JADE.



Sender: r1@192.168.119.1:1099/JADE

Receivers: Owner1@192.168.119.1:1099/JAD

Reply-to:

Communicative act: inform

Content:
N9IjI3KAJ7aeh8jLYwZON8BGnfhcCwoeOlVGPuaiS+UmZ

Language: English

Encoding:

Ontology: Responce

Protocol: Null

Conversation-id:

In-reply-to:

Reply-with:

Reply-by:

User Properties:

Figure 12: Message sniffed from ServiceProvider1 to Owner1.

8. Conclusions

We presented a novel BROadcast based Secure Mobile Agent Protocol (BROSMAP) for real-time distributed applications. The proposed broadcast architecture of MA and hybrid approach of using symmetric and asymmetric cryptography provides distributed applications with efficiency, good performance, and security. The proposed protocol introduces the broadcast architecture for mobile agent systems, where copies of the agent are sent to different service providers simultaneously. It also provides the fundamental security requirements of mutual authentication, authorization, integrity, confidentiality, accountability, and nonrepudiation. BROSMAP was also proven to withstand many attacks such as replay, man in the middle, masquerade, unauthorized access, and modification attacks. Moreover, the broadcast architecture enables the system to continue its functionality in a flawless manner even if one of the agents is maliciously killed or lost. Formal verification test was carried out with the Scyther verification tool, which proved the protocol correctness in providing the different security requirements and protection from the different malicious attacks. Performance analysis was also carried out to study the efficiency of the proposed protocol compared to other related protocols. The conducted performance analysis showed that BROSMAP number of operations is reasonable in relation to the security features it provides. Finally, a scenario of a LBS application that incorporates BROSMAP to secure its communication is implemented with JADE. Implementation with JADE proves the viability of the proposed protocol and ability to provide protection for users.

For the future work, we are planning to enhance the system by incorporating a trust model to help users evaluate the honesty and behavior of service providers. Moreover, BROSMAP will be extended to provide availability and anonymity to cover the seven security requirements. Moreover, another interesting area for research is investigating the possibility of enhancing BROSMAP to cover rational attacks in applications such as service provisioning.

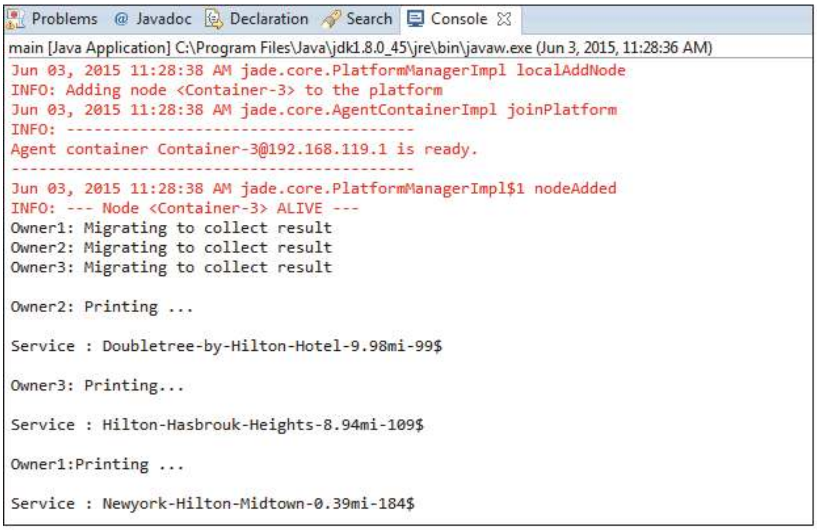
Notations and Definitions

(I)

PLA_i:	Platform i
MA_i:	Mobile agent i
P_i:	Public key of platform i
S_i:	Private key of platform i
R:	Request
OR_i:	Output of result at platform i
K_i:	Session key for platform i
$h(\cdot)$:	Hash value
add_i:	Address of platform i
T_i:	Time stamp generated by platform i
→:	Send
$\|\|$:	Concatenation.

(II)

Owner#i:	Owner platform started in run i
ServiceProvider#i:	Service provider platform contacted in run i
$pk(i)$:	Public key of platform i
$sk(i)$:	Private key of platform i
$R\#i$:	Request generated during run i
OR#i:	Result generated during run i



```
main [Java Application] C:\Program Files\Java\jdk1.8.0_45\jre\bin\javaw.exe (Jun 3, 2015, 11:28:36 AM)
Jun 03, 2015 11:28:38 AM jade.core.PlatformManagerImpl localAddNode
INFO: Adding node <Container-3> to the platform
Jun 03, 2015 11:28:38 AM jade.core.AgentContainerImpl joinPlatform
INFO: --------------------------------------
Agent container Container-3@192.168.119.1 is ready.
--------------------------------------------
Jun 03, 2015 11:28:38 AM jade.core.PlatformManagerImpl$1 nodeAdded
INFO: --- Node <Container-3> ALIVE ---
Owner1: Migrating to collect result
Owner2: Migrating to collect result
Owner3: Migrating to collect result

Owner2: Printing ...

Service : Doubletree-by-Hilton-Hotel-9.98mi-99$

Owner3: Printing...

Service : Hilton-Hasbrouk-Heights-8.94mi-109$

Owner1:Printing ...

Service : Newyork-Hilton-Midtown-0.39mi-184$
```

Figure 13: Final results displayed to user.

$Kir\#i$: Session key generated during run i
$H(\cdot)$: Hash value
$Tj\#i$: Time stamp j generated during run i.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors wish to acknowledge Information and Communication Technology Fund (ICT Fund) for the continued support for the educational development and research. Special thanks go to Ms. Phyllis Burns for proofreading this paper.

References

[1] M. M. de Weerdt, Y. Zhang, and T. Klos, "Multiagent task allocation in social networks," *Autonomous Agents and Multi-Agent Systems*, vol. 25, no. 1, pp. 46–86, 2012.

[2] A. Birukou, E. Blanzieri, and P. Giorgini, "Implicit: a multi-agent recommendation system for web search," *Autonomous Agents and Multi-Agent Systems*, vol. 24, no. 1, pp. 141–174, 2012.

[3] A. Bürkle, A. Hertel, W. Müller, and M. Wieser, "Evaluating the security of mobile agent platforms," *Autonomous Agents and Multi-Agent Systems*, vol. 18, no. 2, pp. 295–311, 2009.

[4] R. Khosla, I. Sethi, and E. Damiani, *Intelligent Multimedia Multi-Agent Systems: A Human-Centered Approach*, vol. 582, Springer, 2013.

[5] C. Y. Yeun, K. Han, D. L. Vo, and K. Kim, "Secure authenticated group key agreement protocol in the MANET environment," *Information Security Technical Report*, vol. 13, no. 3, pp. 158–164, 2008.

[6] N. W. Lo, K.-H. Yeh, and C. Y. Yeun, "New mutual agreement protocol to secure mobile RFID-enabled devices," *Information Security Technical Report*, vol. 13, no. 3, pp. 151–157, 2008.

[7] H. F. Rashvand, K. Salah, J. M. A. Calero, and L. Harn, "Distributed security for multi-agent systems—review and applications," *IET Information Security*, vol. 4, no. 4, pp. 188–201, 2010.

[8] G. Y. Sato, H. J. de Azevedo, and J. A. Barthès, "Agent and multi-agent applications to support distributed communities of practice: a short review," *Autonomous Agents and Multi-Agent Systems*, vol. 25, no. 1, pp. 87–129, 2012.

[9] A. S. Gajparia, C. J. Mitchell, and C. Y. Yeun, "Supporting user privacy in location based services," *IEICE Transactions on Communications*, vol. 88, no. 7, pp. 2837–2847, 2005.

[10] D. M. Konidala, C. Y. Yeun, and K. Kim, "A secure and privacy enhanced protocol for location-based services in ubiquitous society," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '04)*, pp. 2164–2168, IEEE, December 2004.

[11] D. Gavalas, C. Konstantopoulos, K. Mastakas, and G. Pantziou, "Mobile recommender systems in tourism," *Journal of Network and Computer Applications*, vol. 39, pp. 319–333, 2014.

[12] Z. A. Baig, "Multi-agent systems for protecting critical infrastructures: a survey," *Journal of Network and Computer Applications*, vol. 35, no. 3, pp. 1151–1161, 2012.

[13] R. C. Cavalcante, I. I. Bittencourt, A. P. Da Silva, M. Silva, E. Costa, and R. Santos, "A survey of security in multi-agent systems," *Expert Systems with Applications*, vol. 39, no. 5, pp. 4835–4846, 2012.

[14] M. B. Hasan and P. W. C. Prasad, "A review of security implications and possible solutions for mobile agents in e-commerce," in *Proceedings of the Innovative Technologies in Intelligent Systems and Industrial Applications (CITISIA '09)*, pp. 23–29, July 2009.

[15] D. Gollmann, "Computer security," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 5, pp. 544–554, 2010.

[16] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa, "JADE: a white paper," *EXP in Search of Innovation*, vol. 3, no. 3, pp. 6–19, 2003.

[17] F. Bellifemine, F. Bergenti, G. Caire, and A. Poggi, "Jade—a java agent development framework," in *Multi-Agent Programming, Multiagent Systems, Artificial Societies, and Simulated Organizations*, R. Bordini, M. Dastani, J. Dix, and A. El Fallah

Seghrouchni, Eds., vol. 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pp. 125–147, Springer US, Boston, Mass, USA, 2005.

[18] Y. Jung, M. Kim, A. Masoumzadeh, and J. B. D. Joshi, "A survey of security issue in multi-agent systems," *Artificial Intelligence Review*, vol. 37, no. 3, pp. 239–260, 2012.

[19] M. Alzaabi, C. Y. Yeun, and T. A. Martin, "Ensuring anonymity for LBSs in smartphone environment," *Journal of Information Processing Systems*, vol. 7, no. 1, pp. 121–136, 2011.

[20] H. M. N. Al-Hamadi, C. Y. Yeun, M. J. Zemerly, M. A. Al-Qutayri, and A. Gawanmeh, "Formal modeling and verification of DLK protocol," in *Proceedings of the International Conference for Internet Technology and Secured Transactions (ICITST '11)*, pp. 578–583, December 2011.

[21] N. Ahmed and C. D. Jensen, "A mechanism for identity delegation at authentication level," in *Identity and Privacy in the Internet Age*, A. Jsang, T. Maseng, and S. Knapskog, Eds., vol. 5838 of *Lecture Notes in Computer Science*, pp. 148–162, Springer, Berlin, Germany, 2009.

[22] F. Raji and B. Tork Ladani, "Anonymity and security for autonomous mobile agents," *IET Information Security*, vol. 4, no. 4, pp. 397–410, 2010.

[23] W. Kuo, H. Wei, and J. Cheng, "An efficient and secure anonymous mobility network authentication scheme," *Journal of Information Security and Applications*, vol. 19, no. 1, pp. 18–24, 2014.

[24] R. Leszczyna, "Anonymity architecture for mobile agent systems," in *Holonic and Multi-Agent Systems for Manufacturing*, V. Mak, V. Vyatkin, and A. Colombo, Eds., vol. 4659 of *Lecture Notes in Computer Science*, pp. 93–103, Springer, Berlin, Germany, 2007.

[25] C. Garrigues, S. Robles, and J. Borrell, "Securing dynamic itineraries for mobile agent applications," *Journal of Network and Computer Applications*, vol. 31, no. 4, pp. 487–508, 2008.

[26] C.-H. Fong, G. Parr, and P. Morrow, "Security schemes for a mobile agent based network and system management framework," *Journal of Network and Systems Management*, vol. 19, no. 2, pp. 230–256, 2011.

[27] S. Venkatesan, C. Chellappan, T. Vengattaraman, P. Dhavachelvan, and A. Vaish, "Advanced mobile agent security models for code integrity and malicious availability check," *Journal of Network and Computer Applications*, vol. 33, no. 6, pp. 661–671, 2010.

[28] C. Garrigues, N. Migas, W. Buchanan, S. Robles, and J. Borrell, "Protecting mobile agents from external replay attacks," *Journal of Systems and Software*, vol. 82, no. 2, pp. 197–206, 2009.

[29] K. Dhanalakshmi and G. Nawaz, "Matrix hop mobile agent (MHMA) system for e-service applications," *Procedia Engineering*, vol. 30, pp. 1171–1178, 2012, Proceedings of the International Conference on Communication Technology and System Design.

[30] R. Sulaiman and D. Sharma, "Enhancing security in ehealth services using agent," in *Proceedings of the International Conference on Electrical Engineering and Informatics (ICEEI '11)*, pp. 1–6, Bandung, Indonesia, 2011.

[31] H. Guan, H. Zhang, P. Chen, and Y. Zhou, "Mobile agents integrity research," in *Integration and Innovation Orient to E-Society Volume 1. IFIP—The International Federation for Information Processing*, W. Wang, Y. Li, Z. Duan, L. Yan, H. Li, and X. Yang, Eds., vol. 251, pp. 194–201, Springer US, 2007.

[32] G. Geetha and C. Jayakumar, "Data security in free roaming mobile agents," in *Advances in Network Security and Applications, Communications in Computer and Information Science*, D. Wyld, M. Wozniak, N. Chaki, N. Meghanathan, and D. Nagamalai, Eds., vol. 196, pp. 472–482, Springer, Berlin, Germany, 2011.

[33] A. Ouardani, S. Pierre, and H. Boucheneb, "A security protocol for mobile agents based upon the cooperation of sedentary agents," *Journal of Network and Computer Applications*, vol. 30, no. 3, pp. 1228–1243, 2007.

[34] S. Srivastava and G. C. Nandi, "Self-reliant mobile code: a new direction of agent security," *Journal of Network and Computer Applications*, vol. 37, pp. 62–75, 2014.

[35] D. Menacer, H. Drias, and C. Sibertin-Blanc, "Towards a security solution for mobile agents," in *Advances in Information Systems and Technologies, Advances in Intelligent Systems and Computing*, l. Rocha, A. M. Correia, T. Wilson, and K. A. Stroetmann, Eds., vol. 206, pp. 969–979, Springer, Berlin, Germany, 2013.

[36] C. Garrigues, S. Robles, J. Borrell, and G. Navarro-Arribas, "Promoting the development of secure mobile agent applications," *Journal of Systems and Software*, vol. 83, no. 6, pp. 959–971, 2010.

[37] S. Ganapathy, K. Kulothungan, S. Muthurajkumar, M. Vijayalakshmi, L. Yogesh, and A. Kannan, "Intelligent feature selection and classification techniques for intrusion detection in networks: a survey," *EURASIP Journal on Wireless Communications and Networking*, vol. 2013, no. 1, article 271, 2013.

[38] S. Venkatesan and C. Chellappan, "Protection of mobile agent platform through Attack Identification Scanner (AIS) by Malicious Identification Police (MIP)," in *Proceedings of the 1st International Conference on Emerging Trends in Engineering and Technology (ICETET '08)*, pp. 1228–1231, July 2008.

[39] S. Venkatesan, R. Baskaran, C. Chellappan, A. Vaish, and P. Dhavachelvan, "Artificial immune system based mobile agent platform protection," *Computer Standards and Interfaces*, vol. 35, no. 4, pp. 365–373, 2013.

[40] S. Venkatesan and C. Chellappan, "Identifying the split personality of the malicious host in the mobile agent environment," in *Proceedings of the 4th International IEEE Conference Intelligent Systems (IS '08)*, pp. 1440–1444, IEEE, Varna, Bulgaria, September 2008.

[41] T. Kapnoullas, E. Chang, T. Dillon, and E. Damiani, "Security framework for mobile agent platforms (SFMAP)," in *On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops*, R. Meersman and Z. Tari, Eds., vol. 2889 of *Lecture Notes in Computer Science*, pp. 845–858, Springer, Berlin, Germany, 2003.

[42] R. A. Martins, M. E. Correia, and A. B. Augusto, "A literature review of security mechanisms employed by mobile agents," in *Proceedings of the 7th Iberian Conference on Information Systems and Technologies (CISTI '12)*, 4, 1 pages, June 2012.

[43] R. Sulaiman, X. Huang, and D. Sharma, "E-health services with secure mobile agent," in *Proceedings of the 7th Annual Communication Networks and Services Research Conference (CNSR '09)*, pp. 270–277, May 2009.

[44] G. Geetha and C. Jayakumar, "Implementation of trust and reputation management for free-roaming mobile agent security," *IEEE Systems Journal*, vol. 9, no. 2, pp. 556–566, 2014.

[45] T. Olkkonen, "Generic authentication architecture," in *Security and Privacy in Pervasive Computing, Seminar on Network Security, Espoo*, 2006.

[46] K. Han, C. Y. Yeun, T. Shon, J. Park, and K. Kim, “A scalable and efficient key escrow model for lawful interception of IDBC-based secure communication,” *International Journal of Communication Systems*, vol. 24, no. 4, pp. 461–472, 2011.

[47] K. Han, H. Mun, T. Shon, C. Y. Yeun, and J. J. Park, “Secure and efficient public key management in next generation mobile networks,” *Personal and Ubiquitous Computing*, vol. 16, no. 6, pp. 677–685, 2012.

[48] C. Y. Yeun, “Digital signature with message recovery and authenticated encryption (signcryption)—a comparison,” in *Cryptography and Coding (Cirencester, 1999)*, M. Walker, Ed., vol. 1746 of *Lecture Notes in Computer Science*, pp. 307–312, Springer, Berlin, Germany, 1999.

[49] C. Cremers and S. Mauw, *Operational Semantics and Verification of Security Protocols*, Springer, 2012.

[50] C. Cremers, “The scyther tool: verification, falsification, and analysis of security protocols,” in *Computer Aided Verification*, A. Gupta and S. Malik, Eds., vol. 5123 of *Lecture Notes in Computer Science*, pp. 414–418, Springer, Berlin, Germany, 2008.

[51] C. J. Cremers, “Scyther: User manual,” 414–418, 2014.

[52] N. Kahya, N. Ghoualmi, and P. Lafourcade, “Key management protocol in wimax revisited,” in *Advances in Computer Science, Engineering & Applications, Advances in Intelligent Systems and Computing*, D. C. Wyld, J. Zizka, and D. Nagamalai, Eds., vol. 167, pp. 853–862, Springer, Berlin, Germany, 2012.

[53] Y. Lu, X. Wu, and X. Yang, “A secure anonymous authentication scheme for wireless communications using smart cards,” *International Journal of Network Security*, vol. 17, no. 3, pp. 237–245, 2015.

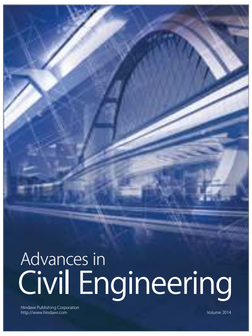
Journal of
Engineering

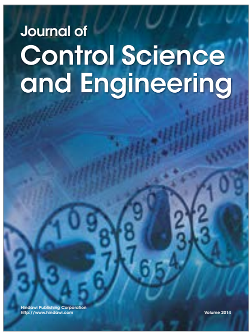
The Scientific World Journal

International Journal of
Rotating Machinery

Journal of
Sensors

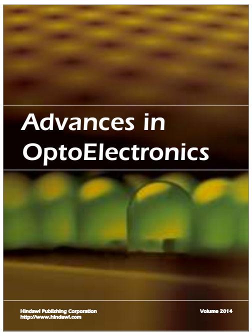
International Journal of
Distributed Sensor Networks


Advances in
Civil Engineering


Journal of
Control Science and Engineering

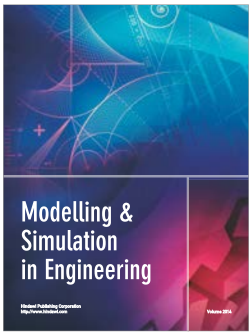
Journal of
Robotics

Journal of
Electrical and Computer Engineering


Advances in
OptoElectronics

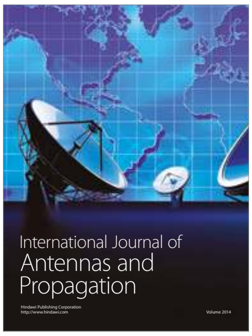
VLSI Design

International Journal of
Navigation and Observation

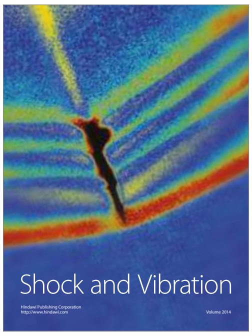

Modelling & Simulation in Engineering

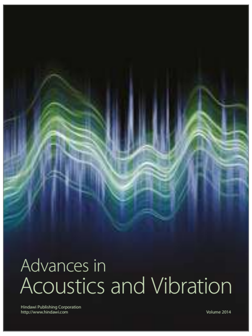
International Journal of
Aerospace Engineering

International Journal of
Chemical Engineering


International Journal of
Antennas and Propagation

Active and Passive
Electronic Components


Shock and Vibration


Advances in
Acoustics and Vibration