Jurnal
Teknologi

Full Paper

A LIGHTWEIGHT ONE-PASS AUTHENTICATION MECHANISM FOR AGENT COMMUNICATION IN MULTI-AGENT SYSTEM BASED APPLICATIONS

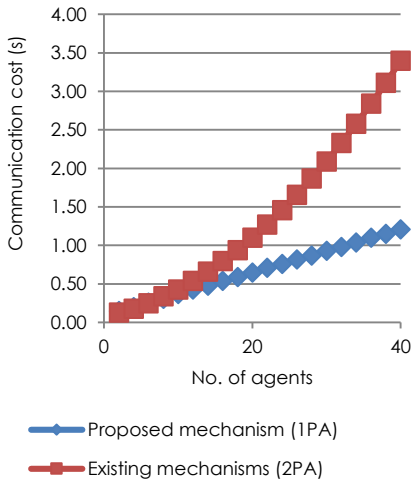
Olumide Simeon Ogunnusi, Shukor Abd Razak, Abdul Hanan Abdullah

Department of Computer Science, Faculty of Computing, Universiti Teknologi Malaysia, 81310 UTM Johor Bahru, Johor, Malaysia

Article history
Received
15 May 2015
Received in revised form
1 July 2015
Accepted
11 August 2015

*Corresponding author
shukorar@utm.my

Graphical abstract



Abstract

The social nature of mobile agent and its ability to carry its principal's confidential information necessitate the need to secure its communication with other agent(s) in an agent system. Most importantly, an agent communication security mechanism must be able to prevent unknown or visiting agent from participating in legitimate agent communication. Most of such mechanisms adopt two-pass authentication technique without due consideration of the enormous overheads generated by the mechanisms. These overheads are more noticeable in multi-agent system based applications with large number of agents such as smart grid. The main focus of this paper therefore, is to design a lightweight mechanism for agent communication confidentiality protection in a local area network (LAN) or intranet using one-pass authentication approach. The proposed mechanism adopted both symmetric and asymmetric cryptosystems to protect agent certificate transmission between task agents and agent execution host. The results show that the memory utilization, communication and computation costs of the proposed mechanism are remarkably lower than that of the two-pass authentication based mechanisms.

Keywords: One-pass authentication, two-pass authentication, agent communication, confidentiality protection

1.0 INTRODUCTION

Confidentiality protection in agent communication is bothered on ascertaining that no external agent takes part in agent communication or have access to the exchanged information [1, 2]. This could be achieved in two ways: one is by isolating the external agent to a neutral host and deprive it from communicating with legitimate agents in other hosts; two, is by applying a strong cryptographic authentication scheme to fence out and kill external agent with fake identity. JADE fundamentally secured agent communication channel (ACC) using secured socket layer (SSL) but deficient in preventing man-in-the-middle (MITM) attack. In the proposed security model, authenticity is established using digital signature, agent certification, and cryptographic authentication of certificates.

Rossebo and Brœk [3] identified two techniques that could be used for agent authentication: *one-pass authentication and two-pass authentication techniques*. One-pass authentication pattern involves communication between two parties, in this study, between task agents and execution host in the form of one-way asynchronous message passing for agent identity establishment. This is referred to as unilateral one-pass authentication approach. Conversely, two-pass authentication involves mutual communication between a pair of entities such as agent pair. This study has adopted the combination of symmetric and

asymmetric cryptographic schemes to facilitate the privacy of certificate transmission between task agents and execution host. While using the combined cryptosystems, the research is focusing on the best manner to minimize the overheads the propose security mechanism impose on the host network.

The rest of the paper is organized as follows: Section 2 presents the related work while the proposed security mechanism is described in Section 3. In Section 4, the experimental procedure is discussed. The comparative cost analysis of existing and proposed mechanisms are presented and discussed in Section 5 while Section 6 gives the implementation details with the comparative analysis of results. Section 7 concludes the paper and presents future work.

2.0 RELATED WORK

Confidentiality threat to agent communication is a treat against the privacy of the information shared or exchanged by mobile agents. For instance, the shared information may be confidential information about the agents' principals. Agents participating in the exchange of such private information must be able to establish the identity of the peer agent with whom to exchange the information using strong cryptographic authentication scheme. The existing mechanisms [Xu, Zhu [4]; Guo, Chang [5]; Ben Ameur, Zarai [6]] adopt two-pass authentication approach to establish the identity of entities before communication is permitted between them.

In two-pass authentication technique, each agent must possess the certificates of all other agents taking part in the communication. Contrarily, only the authenticating platform (host) is required to possess the certificates of all the agents while each agent carries only its certificate in the case of one-pass authentication method. Some agent system security approaches enforce both internal and external security. For instance JADE-S add-on [7] for the JADE agent platform gives the user the privilege to restrict access to the services of the platform using access control lists and user authentication. While this method addresses authentication in the administrative aspects of platform management, it lacks the mechanism to assist authentication for agent interaction and services [1].

Sulaiman and Sharma [8] and Sulaiman, Huang [9] proposed a multi-agent based security mechanism (MAgSeM) that is used to improve a traditional non-agent based system. The authors claimed that as a result of the interactive, autonomous, extensible and mobile properties of the agents, the agents were able to perform their tasks with minimal interaction with the user. The key to decipher information is kept with the sender. A token is sent to the receiver to sign and forward it back to the sender to receive the key to decipher the information. In this security mechanism, the sender is in control of the transferred information while the details of the decryption are unknown to the receiver. The authors, however, assume that the communicating agents exchanged certificates via a secure channel. The type of technology adopted for agent certification is not specified.

3.0 PROPOSED MECHANISM

The focus of the proposed mechanism is on the confidentiality protection of agent communication in a multi-agent system running on LAN or intranet. The design of the mechanism extends the key distribution protocol phases of [10] to 5 phases as shown in Appendices A-1 and A-2.

Phase I - Pre-configuration phase: The security administrator configures the local keystores and certificate stores of agent server (AS) and execution host (EH) which are respectively used for the storage of public/private key pairs and digital certificates.

Phase II - Initialization phase: At this phase, the local area network (LAN) or intranet was disconnected from the internet so as to avert the possibility of external agent participating in the initialization process. The reconnection of the LAN/intranet to internet can be established after this phase. The activities at this phase are creation of task agents, generation of RSA keys etc.

Phase III - Appointment and registration phase: Appointment of EH and the registration of both EH and other network host(s) take place.

Phase IV - Encryption, Agent certificate signing and AS signature hashing phase: This phase covers the secret key encryption, agent certificate signing and hashing of AS signature.

Phase V - Agent mobility and authentication phase: At this phase, the task agents are transmitted from the agent server to the agent execution host and their identities are verified.

The notational algorithm of the proposed security mechanism is depicted in Figure 1 while the meanings of the notations are presented in Table 1.

1. SA: **Configure** AS_{ks}/AS_{cs} & EH_{ks}/EH_{cs}
2. SA: AS ← **Request**[$AS(k_i, k_j)$]
3. AS: AC ← **Generate**[$AS(k_i, k_j)$]
4. SA: CA ← **Request**[(AS_{cert}); $AS(k_i, k_j)$]
5. CA: AS ← [(AS_{cert}); $AS(k_i, k_j)$]
6. SA: AC ← **Request**[Create(TA)]
7. AC: AS ← **Create**(TA)
8. SA: AS ← **Request**[$TA(k_i, k_j)$]
9. AS: AC ← **Generate**[$TA(k_i, k_j)$]
10. AC: CA ← [**Request**(TA_{cert}); $TA(k_i, k_j)$]
11. CA: AS ← [**Generate**(TA_{cert}); $TA(k_i, k_j)$]
12. AC: TA_{ks} ← $TA(k_i, k_j)$
13. AC: EH_{cs} ← EH_{cert}
14. SA: AS ← **Request**[Generate(TA_{sk})]
15. AS: AC ← **Generate**(TA_{sk})
16. AS: TA ← **Load**[$TA_{cert+(ki,kj)}$+TA_{sk}]
17. AS: Repeat steps 6 to 16 until the req. max. no. of agents is reached.
18. SA: AS ← **Request**[Appoint(EH)]
19. AS: **Appoint**(EH)
20. SA: AS ← **Request**[$EH(k_i, k_j)$]
21. AS: AS ← **Generate**[$EH(k_i, k_j)$]
22. AS: CA ← [**Request**(EH_{cert}); $EH(k_i, k_j)$]

23. CA: AS ← [Generate(EHcert); EH(ki, kj)]
24. AS: EH ← [EHcert; EH(ki, kj)
25. EH: EH_{ks} ← $EH(k_i, k_j)$
26. EH: EH_{cs} ← EH_{cert}
27. SA: AS ← **Request**[(Encrypt(TA_{cert}),sk)]/* sk is the secret key */
28. AS: AC ← [Request[(Encrypt(TA_{cert}),sk)]]
29. AC: AS ← **Encrypt**[(TA_{cert}),sk]
30. SA: AS ← **Request**[sign(TA_{cert}, $AS(k_i)$)]
31. AS: AC ← [**Request** [sign(TA_{cert}, $AS(k_i)$)]]
32. AC: AS ← **sign**(TA_{cert}, $AS(k_i)$)
33. AS: Repeat lines 27-32 for all deployed task agents
33. SA: AS ← **Request**[hash(AS_{sign})]
34. AS: AC ← [Request[hash(AS_{sign})]]
35. AC: AS ← **hash**(AS_{sign})]
36. SA: AS ← **Request**[Encrypt(sk), $EH(k_j)$]
37. AS: AS ← **Encrypt**(sk, $EH(k_j)$)
38. AS: EH ← [Encrypt(sk, $EH(k_j)$)]
39. SA: AS ← **Request**[TA ← (Encrypt(TA_{cert}))]
40. AS: EH ← [TA ← (Encrypt(TA_{cert}))]
41. AS: Repeat lines 39 & 40 for all deployed task agents
41. SA: EH ← **Request**[hash(AS_{sign}); Compare(h_d // h_s)]
42. SA: EH ← **Request**[Decrypt((sk), $EH(k_i)$)]
43. EH: ks ← **Decrypt**((sk), $EH(k_i)$/ *ks is the key store*/
42. EH: cs ← **Decrypt**((TA_{cert}), sk) / *cs is the certificate store*/
43. EH: **end**

Figure 1 Notational algorithm of the proposed mechanism

Table 1 Igorithm notation and meaning

Notation	Meaning
SA	Security administrator
AS	Agent server
CA	Certificate authority
EH	Execution Host
TA	Task agent
k_i	Private key
k_j	Public key
ks	Key store
cs	Certificate store
Cert.	Certificate
sk	Secret key
AS_{sign}	AS signature
h_d	Derived hash value
h_s	Sent hash value

4.0 EXPERIMENTAL PROCEDURE

When the network was successfully set up, the agent server encrypts agent certificate with AES secret key and signed the encrypted certificate before transmission to the execution platform. It also hashes the signature with 160-bit SHA-1 algorithm and sends the hash value and the original signature to the execution host for authentication. The secret key was encrypted with RSA public key of execution host before it was sent to the execution host. On the arrival of agent and secret key at the execution host, the execution host hashes the agent server signature using the same hash function (160-bit SHA-1) and compares the derived hash value with that sent by the agent server. Thereafter, execution host decrypts the secret key with its private key and use the secret key to decrypt the agent certificate. In this way, we ensure that no unauthorized agent gained entrance into the execution platform to exercise man-in-the-middle attack on private communication among the legitimate task agents thus we ensure confidentiality of agent communication based on identity verification.

5.0 COMPARATIVE COST ANALYSIS OF EXISTING MECHANISMS WITH OUR PROPOSED MECHANISM

In this study, the costs of one-pass agent authentication technique are measured in terms of memory utilization, communication and computation costs and the results are compared with that of two-pass agent authentication approach. For two-pass agent authentication technique, each task agent participating in agent communication must possess the certificates of all other task agents. Appendices B-1 and B-2 respectively depict the notations used for one-pass and two-pass authentication techniques and their corresponding meanings. Three storage locations are of importance here: the agent server, task agent keystore and the agent execution host. The aggregate memory utilized $\boldsymbol{MO_{2T}}$ was derived from the following expression:

$$MO_{2T} = MO_{2AS}+MO_{2TA}+MO_{2EH} \quad (1)$$

The memory consumed at the execution host, agent keystore, and agent server are respectively $\boldsymbol{MO_{2EH}}$, $\boldsymbol{MO_{2TA}}$, and $\boldsymbol{MO_{2AS}}$. The storage utilized at each location is presented in Eq. (2) to (4).

$$MO_{2AS}=n*(M_{sta/c}+M_{pkta/as}+M_{kta/as})+½*n*M_{ska}+M_{sas}+M_{k/as} \quad (2)$$
$$MO_{2TA}=n^{2}*(M_{sta}+M_{pk/ta}+½*M_{sk/ta})+n*M_{k/ta}+n*M_{s/hv} \quad (3)$$
$$MO_{2EH}=M_{aspk}+M_{k/eh}+M_{pk/eh} \quad (4)$$

Similarly, the time for communication between two entities was also tracked and the aggregate communication time $\boldsymbol{COM_{2T}}$ was derived as shown in Eq. (5).

$$COM_{2T}=COM_{2AS\text{-}CA}+COM_{2AS\text{-}EH}+COM_{2AS\text{-}TA}+COM_{2TA\text{-}TA} \quad (5)$$

The time for communication between certificate authority and agent server, between agent server and execution host, between agent server and task agent, and between agent pair are respectively $\boldsymbol{COM_{2AS\text{-}CA}}$, $\boldsymbol{COM_{2AS\text{-}EH}}$, $\boldsymbol{COM_{2AS\text{-}TA}}$, *and* $\boldsymbol{COM_{2TA\text{-}TA}}$. The communication cost for each of the Eq. (5) components was determined using similar mathematical expressions in [11] as shown in Eq. (6) to (9).

$$COM_{2AS\text{-}CA}=T_{rcas}+T_{gsca}+T_{as/ca}+T_{ca/as}+n*(T_{rac}+T_{ta/as} \quad (6)$$
$$COM_{2AS\text{-}EH} = T_{skh}+T_{ehk/eh}+T_{c/eh}+n*T_{dta} \quad (7)$$
$$COM_{2AS\text{-}TA} = n*(T_{tac/ta} + T_{sk/tas} + T_{k/tas}) \quad (8)$$
$$COM_{2TA\text{-}TA}=n*(n- 1)*(T_{ac/ta}+T_{esk/ta}+T_{ta/p}+T_{s/hv/ta}) \quad (9)$$

In the same manner, computation activities take place at the agent server and the execution host. The computation time was also tracked at each of these units and the aggregate computation cost $\boldsymbol{CO_{2T}}$ was obtained from the following expression:

$$CO_{2T} = CO_{2AS}+CO_{2EH} \quad (10)$$

Where CO_{2AS} is the computation cost at the agent server while the computation cost at the execution host is CO_{2EH}. Each of the components' computation costs is derived using the expressions below:

$$CO_{2AS} = T_{ask}+T_{ehk}+T_{ska}+T_{esa}+T_{hcs}+n*(T_{tak}+T_{scak}) \quad (11)$$
$$CO_{2EH} = n(n-1)*(T_{ta/h}+T_{dska}+T_{dca}) \quad (12)$$

For one-pass agent authentication, message is transmitted by the task agents by sending their certificates to the execution host for authentication.

Equations similar to Eq. (1), (5), and (10) were also used to compute the memory utilization, communication and computation costs respectively as shown below.

$$MO_{1T} = MO_{1AS}+MO_{1TA}+MO_{1EH} \quad (13)$$

Each component of Eq. 13 was derived using Eq. (14), (15), and (16)

$$MO_{1AS} = M_{sas}+M_{ask}+M_{asp}+n*M_{sta} \quad (14)$$
$$MO_{1TA} = n*(M_{sta}+M_{k/ta}) \quad (15)$$
$$MO_{1EH}=n*M_{sta}+M_{ska}+M_{aspk}+M_{s/hv}+M_{k/eh}+M_{pk/eh} \quad (16)$$

Eq. 17 was used for the computation of communication cost whose components give the time for communication between two entities as presented below.

$$COM_{1T}=COM_{1AS-CA}+COM_{1AS-EH}+COM_{1AS-TA}+COM_{1TA} \quad (17)$$

Thus,

$$COM_{1AS-CA}=T_{rcas}+T_{gsca}+T_{as/ca}+T_{ca/as}+n*(T_{rac}+T_{ta/as} \quad (18)$$
$$COM_{1AS-EH}=n*T_{dta}+T_{sk/aeh}+T_{skh}+T_{ehc/aeh}+T_{ssh}+T_{aehk/aeh} \quad (19)$$
$$COM_{1AS-TA} = n*T_{tac/ta} \quad (20)$$
$$COM_{1TA-EH} = n*T_{tac/eh} \quad (21)$$

To compute the computation cost of the proposed mechanism, the computation activities at the agent server and the execution host were aggregated as expressed in Eq. 22.

$$CO_{1T} = CO_{1AS}+CO_{1EH} \quad (22)$$

We have,

$$CO_{1AS} = T_{ask}+T_{ehk}+T_{ska}+T_{esa}+T_{hcs}+n*(T_{tak}+T_{scak}) \quad (23)$$
$$CO_{1EH} = T_{dskt}+n*(T_{esk}+T_{cs}) \quad (24)$$

6.0 IMPLEMENTATION DETAILS WITH COMPARATIVE ANALYSIS OF RESULTS

Java programming language and Java Agent Development Framework (JADE) [[7, 12, 13]] were used to implement and test the proposed mechanism. JADE is an agent system development framework aimed at developing multi-agent systems and agent based applications conforming to Foundation for Intelligent Physical Agents (FIPA) standards. The application was tested on a LAN with a server having Intel Core i5 processor with 2.40GHz speed, 4GB RAM and Windows Ultimate (64 bits) operating system with Oracle virtualbox running Ubuntu Linux operating system (1GB RAM) used for the execution environment for the task agents. The application is made up of two program modules running on two different JADE platforms connected by a local area network. Linux box is the agent execution environment where the task agents run while the Agent Controller and certificate authority agent run on the Windows machine. During the experiment, the task agents are sent to the execution host one after the other while there are sufficient resources at the execution host. The main JADE container resides on agent server while other hosts have JADE platform running on them.

6.1 Results Analysis

Analysis of the performance of the mechanism implemented on two-pass agent authentication technique and the proposed mechanism are carried out in terms of memory utilization, communication and computation costs of providing agent communication confidentiality protection as illustrated in Figures 2, 3, and 4. In this section, we verify our mechanism performance by implementation results. Both the proposed and existing mechanisms were executed 5 times for every number of agents considered and analyses of the memory utilization, communication and computation costs were performed. In all, 40 agents were considered starting from 2 being the minimum number of agents in a multi-agent system. The communication cost was regarded as the average time required for an entity to send message to another entity. Similarly, the computation cost is the average time a processor devoted for each sub-operation such as encryption, key generation, hash value comparison, decryption, digital signature etc.

1) Memory utilization

The average storage utilized by the existing and proposed mechanisms are measured respectively using Eqs. (1) and (13). Figure 2 depicts the memory utilization comparison of the two mechanisms. From the plot, it can be observed that the existing mechanisms use more storage which becomes more and more noticeable as the number of agents increases. This implies that the existing mechanism incurred greater memory overhead compared to the proposed mechanism.

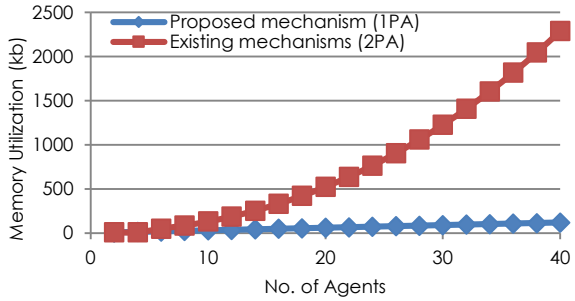


Figure 2 Comparison of confidentiality verification storage Utilization

2) Communication cost

The communication time required to provide confidentiality protection for agent communication is analyzed in Section 5 while Eqs. (5) and (7) present the mathematical models used to compute the time for communication between two entities for every number of agents considered. This is further expressed graphically in Figure 3. The Plot shows that the existing mechanism takes higher communication time to provide agent communication confidentiality protection compared to the communication time needed by the proposed mechanism. These mechanisms were implemented and tested on the system with configuration of 4GB RAM and 2.4GHz Intel Core i5 processor.

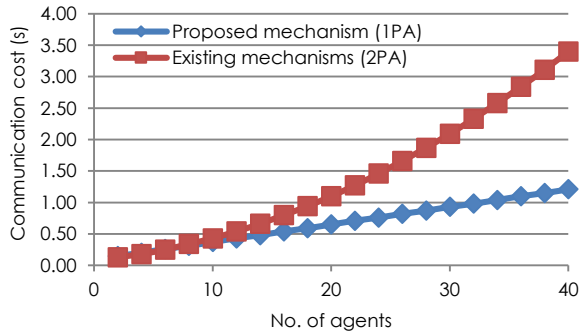


Figure 3 Comparison of confidentiality verification Communication cost

3) Computation cost

The computation cost of providing confidentiality protection for agent communication is also analyzed in Section 5 for both the existing and the proposed mechanisms. The computation costs for the existing and proposed mechanisms were determined using Eq. (10) and (22) respectively. Figure 4 shows the comparative aggregate computation costs needed to provide agent communication confidentiality protection for each number of agents injected into the network due to key generation, asymmetric and symmetric encryption, hashing, and signing of certificates. It could be observed in Figure 4 that the magnitude of the computation cost differential is minimal with small number of agents but it becomes more prominent as the number of agents increases. For example, 40 agents with computation cost difference of 18.84s in favour of the proposed mechanism with 49.3% improvement over the existing mechanisms. Consequently, as the number of agents injected into the network increases, the cost gain of the proposed mechanism over the existing one, in terms of computational complexity, is more evident.

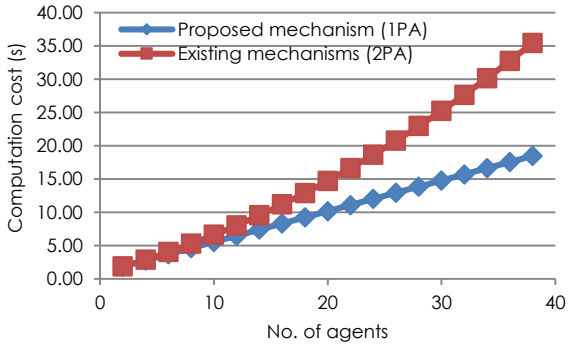


Figure 4 Comparison of confidentiality verification Computation complexity

7.0 CONCLUSION AND FUTURE WORK

This paper presents a one-pass authentication technique for agent communication confidentiality protection against intrusion and its attendant private information leakage to third party. Our proposed mechanism was evaluated on the basis of memory utilization, communication and computational complexities and compared with the existing mechanisms that adopt two-pass authentication approach. The results show that the proposed mechanism performed better as shown in Figures 2, 3, and 4. Our mechanism is motivated from centralized agent authentication approach as a measure to reduce overheads on network. However, instituting a parallel or concurrent authentication of agents on the execution platform and establishing fault tolerance defense for the platform, we give a new direction as it will further reduce the overheads the mechanism imposes on the host network and enhances its reliability.

Acknowledgement

This research is supported by Universiti Teknologi Malaysia through grant: Q.J130000.2513.08H30 and Federal Polytechnic, Ado-Ekiti, Nigeria.

References

[1] Braubach, L., K. Jander, and A. Pokahr. 2013. *A Practical Security Infrastructure for Distributed Agent Applications*, in *Multiagent System Technologies*. Springer. 29-43.

[2] Stallings, W. 2007. *Network Security Essentials: Applications and Standards*. Pearson Education India.

[3] Rossebo, J. E. and R. Brœk. *2006.* Towards a Framework of Authentication and Authorization Patterns for Ensuring Availability in Service Composition. In *Availability, Reliability and Security, ARES 2006. The First International Conference on*. IEEE.

[4] Xu, J., W.-T. Zhu, and D.-G. Feng. 2011. *An* efficient Mutual Authentication and Key Agreement Protocol Preserving User Anonymity in Mobile Networks. *Computer Communications*. 34(3): 319-325.

[5] Guo, C., C.-C. Chang, and C.-Y. Sun. 2013. *Chaotic* Maps-Based Mutual Authentication and Key Agreement Using Smart Cards for Wireless Communications. *Journal of Information Hiding and Multimedia Signal Processing*. 4(2): 99-109.

[6] Ben Ameur, S., *et al.* 2014. A Lightweight Mutual Authentication Mechanism for Improving Fast PMIPV6-Based Network Mobility Scheme. In *Network Infrastructure and Digital Content (IC-NIDC), 2014 4th IEEE International Conference on*. IEEE.

[7] Vila, X., A. Schuster, and A. Riera. 2007. Security for a Multi-Agent System based on JADE. *Computers & Security*. 26(5): 391-400.

[8] Sulaiman, R. and D. Sharma. 2011. Enhancing Security in E-Health Services Using Agent. In *Electrical Engineering and Informatics (ICEEI), 2011 International Conference on*. IEEE.

[9] Sulaiman, R., X. Huang, and D. Sharma. 2009. E-Health Services with Secure Mobile Agent. In *Communication Networks and Services Research Conference, 2009. CNSR'09. Seventh Annual*. IEEE.

[10] Ismail, L. and E. Barka. 2008. Key Distribution Framework for a Mobile Agent Platform. In *Next Generation Mobile Applications, Services and Technologies, 2008. NGMAST'08. The Second International Conference on*. IEEE.

[11] Srivastava, S. and G. Nandi. 2014. Self-Reliant Mobile Code: A New Direction of Agent Security. *Journal of Network and Computer Applications.* 37: 62-75.

[12] Board, J. 2005. *Jade Security Guide.* JADE-S Version, 2.

[13] Bellifemine, F. *et al*. 2002. *Jade Programmer's Guide.* Jade version, 3.

APPENDIX A

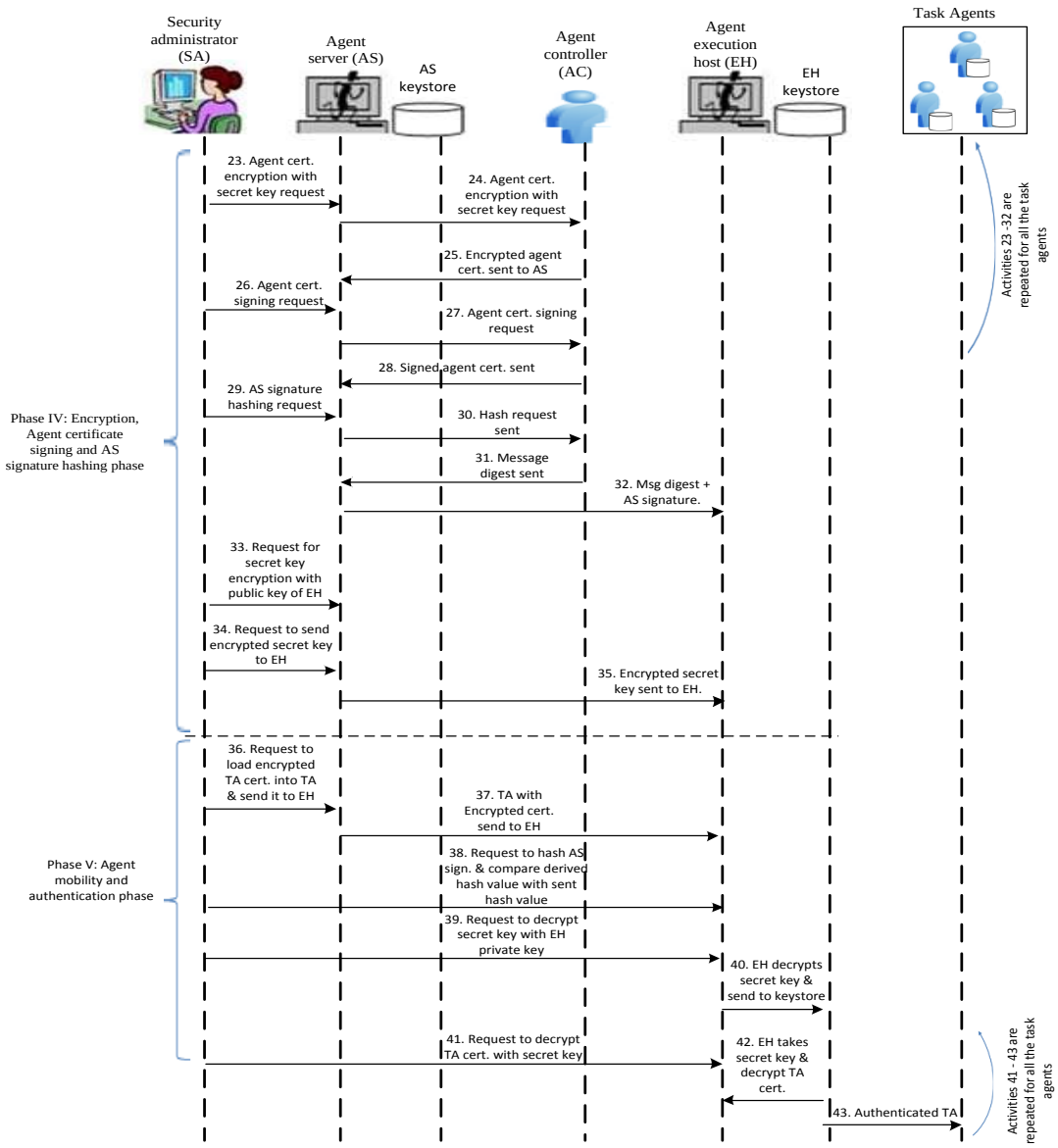


Figure A1 Phases I, II and III of the proposed mechanism

APPENDIX B

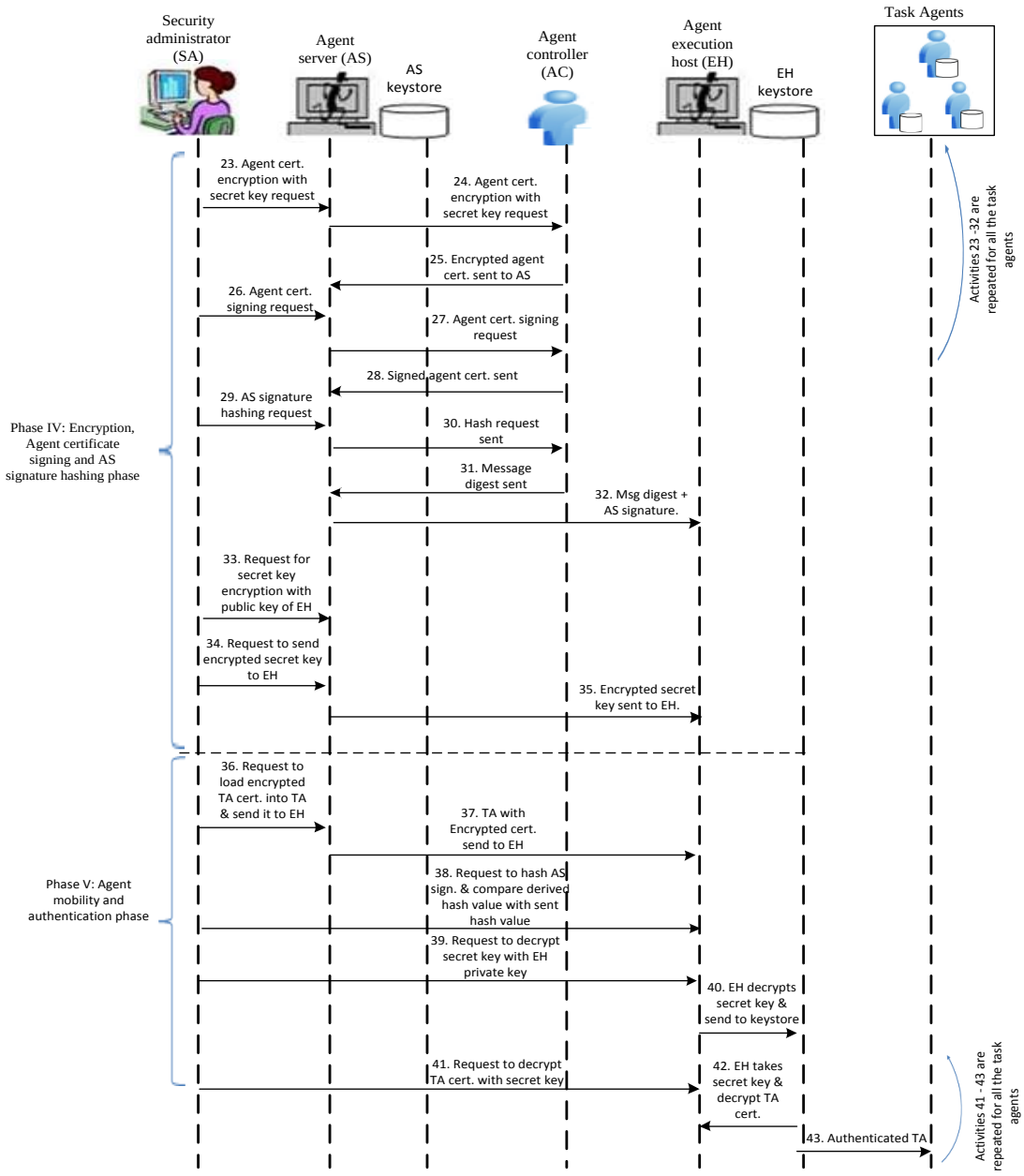


Figure B1 Phases IV and V of the proposed mechanism

APPENDIX C

Table C1 Notations used in the expressions for one-pass agent authentication technique

Notation	Meaning	Notation	Meaning
M_{sas}	Memory for storing AS certificate with its public key in AS.	T_{ska}	Time to generate secret key for TA.
M_{sta}	Memory for storing the certificate of a task agent.	T_{esa}	Time to encrypt secret key in TA with public key of EH.
M_{ska}	Memory for storing the secret key of a task agent.	T_{scak}	Time to sign the cert. of TA with private key of AS.
M_{ask}	Memory for storing public key of AS.	T_{dskt}	Time to decrypt secret key of TA with EH private key.
M_{asp}	Memory for storing private key of AS.	T_{cs}	Time to decrypt TA cert. with secret key.
$M_{s/hv}$	Memory for storing AS signature and hash value sent by AS to EH.	T_{esk}	Time for EH to hash original sign. of AS and comp. derived hash value with the hash value sent by AS.
T_{rcas}	Time taken for AS to request for its own certificate from CA.	T_{ask}	Time to generate RSA keys for AS.
T_{gsca}	Time taken for CA to send AS certificate to AS.	T_{ehk}	Time to generate RSA keys for EH.
$T_{as/ca}$	Time taken for AS to request for EH certificate from CA.	T_{tak}	Time to generate RSA keys for TA.
$T_{ca/as}$	Time taken for CA to send EH certificate to AS.	T_{hcs}	Time to hash signature of AS in TA cert.
T_{rac}	Time taken for AS to request for a TA certificate from CA.	T_{dta}	Time taken for AS to transmit TA to EH.
$T_{ta/as}$	Time taken for CA to send a TA certificate to AS.	n	No. of task agents injected into the network
T_{skh}	Time taken for AS to send its public key to EH.		
T_{ssh}	Time taken for AS to send its signature & hash value to EH.		
$T_{sk/aeh}$	Time taken for AS to securely transmit secret key to EH.		
$T_{ehc/aeh}$	Time taken for AS to send EH certificate to EH.		
$T_{aehk/aeh}$	Time taken for AS to send EH public/private key pair to EH.		

Table C2 Notations used in the expressions for two-pass agent authentication technique

Notation	Meaning	Notation	Meaning
M_{sas}	Memory for AS certificate with its public key in AS.	$M_{k/eh}$	Mem. for public key of EH.
$T_{ehk/eh}$	Time for AS to securely send EH public/private key pair to EH.	$M_{pk/eh}$	Mem. for private key of EH.
$T_{esk/ta}$	Time to transmit encrypted secret key from one agent to another.	M_{ska}	Mem. of AS for the secret key of a task agent.
$M_{k/ta}$	Mem. for public key of a task agent in each task agent keystore.	T_{dta}	Time for AS to transmit a task agent to EH.
$M_{pk/ta}$	Mem.for private key of a task agent in each task agent keystore.	T_{skh}	Time for AS to send its public key to EH.
$M_{sk/ta}$	Mem. for the secret key of a task agent in each task agent keystore	M_{ask}	Mem. for public key of AS.
$M_{s/hv}$	Mem. for AS signature and hash value in each task agent	$M_{cta/as}$	Mem. for certificate of task agent in AS.
$M_{sta/c}$	Mem. for a task agent certificate in each agent certificate store.	$M_{pkta/as}$	Mem. for public key of task agent in AS .
T_{isha}	Time for AS to send its signature and hash value to a task agent in EH.	$M_{pk/as}$	Mem. for AS private key in its keystore.
$T_{tac/ta}$	Time for AS to insert task agent certificate into TA	$T_{ta/h}$	Time taken by TA to hash signature of another TA and compare the derived hash value with the hash value sent by the agent.
$T_{sk/tas}$	Time for AS to insert secret key into task agent keystore	T_{dca}	Time for TA to decrypt its certificate using the secret key.
$T_{k/tas}$	Time for AS to insert public/private key pair into task agent keystore.	T_{dska}	Time taken for TA to decrypt the secret key in its certificate with its private key.
$T_{ac/ta}$	Time for each task agent to send its certificate to another task agent	T_{rcas}	Time taken for AS to request for its own certificate from CA.
$T_{ta/p}$	Time for task agent to send its public key to another task agent.	$T_{s/hv/ta}$	Time for a task agent to send its signature and hash value to another task agent.
		n	No. of TAs injected to the network.