Contributions to the security of mobile agent systems

Hind Idrissi

Université Mohammed V
Faculté des Sciences
Rabat

La Rochelle
UNIVERSITÉ

N° d'ordre : 2891

THÈSE DE DOCTORAT

Présentée par

Hind IDRISSI

Discipline : Informatique
Spécialité : Informatique et Sécurité de l'Information

Titre

Contributions à la Sécurité des Systèmes d'Agents Mobiles

Soutenue le **Vendredi 15 Juillet 2016**

Devant le jury

Président :		
Mr. Said EL HAJJI	PES	Faculté des Sciences, Université Mohammed V de Rabat, Maroc
Examinateurs :		
Mr. Arnaud REVEL	PU	Université de La Rochelle, France
Mr. El Mamoun SOUIDI	PES	Faculté des Sciences, Université Mohammed V de Rabat, Maroc
Mr. Charly POULLIAT	PU	INP-ENSHEEIT, Toulouse - France
Mme. Lucile SASSATELLI	MC	Université Nice Sophia Antipolis, Nice - France
Mr. Jalal LAASSIRI	PH	Faculté des Sciences, Université Ibn Tofail de Kenitra, Maroc

Université Mohammed V
Faculté des Sciences
Rabat

UNIVERSITÉ La Rochelle

N° d'ordre : 2891

DOCTORAL THESIS

Presented By

Hind IDRISSI

Option : Computer Sciences
Discipline : Computer Sciences and Information Security

Title

Contributions to the Security of Mobile Agent Systems

Defended on **July** 15^{th}**, 2016**

Jury

President :		
Mr. Said EL HAJJI	PES	Faculty of Sciences, Mohammed-V University in Rabat, Maroc
Reporters :		
Mr. Arnaud REVEL	PU	University of La Rochelle, France
Mr. El Mamoun SOUIDI	PES	Faculty of Sciences, Mohammed-V University in Rabat, Maroc
Mr. Charly POULLIAT	PU	INP-ENSHEEIT, Toulouse - France
Mme. Lucile SASSATELLI	MC	Nice Sophia Antipolis University, Nice - France
Mr. Jalal LAASSIRI	PH	Faculty of Sciences, Ibn Tofail University in Kenitra, Maroc

Faculté des Sciences, 4 Avenue Ibn Battouta B.P. 1014 RP, Rabat - Maroc
Tel +212 (05) 537 77 18 34/35/38, Fax : +212 (05) 537 77 42 61, http ://www.fsr.ac.ma

Avant-Propos

Cette thèse a été réalisé en cotutelle dans le cadre du Projet Hubert Curien Volubilis N°. **27041QL**.

Les travaux présentés dans ce mémoire ont été effectués au Laboratoire de Mathématiques, Informatique et Application (LabMIA) de la Faculté des Sciences de Rabat (FSR), Université Mohammed V de Rabat au Maroc sous la direction du professeur El Mamoun SOUIDI, et au Laboratoire Informatique, Image et Interaction (L3I), Université de La Rochelle en France sous la direction du professeur Arnaud REVEL, dans le cadre d'une thèse en cotutelle.

Je commencerai par présenter ma plus vive gratitude à mon Directeur de thèse M. El Mamoun SOUIDI, Professeur d'Enseignement Supérieur à la FSR. Grâce à ses encouragements et ses conseils qui m'a procuré, il a su me mettre sur le chemin afin de mener à bien mes travaux de recherche. Je ne saurais exprimer ma profonde gratitude à son égard et l'ultime respect que je lui porte.

Je veux aussi exprimer toute ma reconnaissance à mon directeur de thèse en France, M. Arnaud REVEL, Professeur des Universités à l'Université de La Rochelle. Je tiens à le remercier profondément de m'avoir intégrée au sein de l'équipe « Interactivité et Dynamique des Systèmes » du laboratoire L3I, et de m'avoir consacré son temps et son énergie malgré ses préoccupations et son emploi du temps très chargé. Sa motivation, ses encouragements et ses conseils m'ont été d'une très grande aide pour mener à terme ce travail.

Je souhaite remercier profondément M. Said EL HAJJI, Professeur d'Enseignement Supérieur à la FSR, directeur du laboratoire LabMIA d'avoir accepté de présider le jury de ma thèse. Je tiens à lui présenter ma gratitude de m'avoir intégrée au sein de son laboratoire et d'avoir toujours été présent pour ses doctorants avec ses conseils scientifiques et humains, malgré ses occupations.

Mes remerciements vont à Mr. Charly POULLIAT, Professeur des Universités à l'INP-Ecole Supérieure Electronique Electrotechnique Informatique Hydraulique Télécommunications (INP-ENSHEEIT), Toulouse-France, pour avoir accepté de rapporter mon travail et de participer au jury. Ses remarques, ses conseils et ses suggestions m'apporterais de grand supports.

Je remercie également Mr. Jalal LAASSIRI, Professeur d'Habilité à la FS-Université Ibn Tofail de Kenitra, pour avoir accepté de rapporter ce travail et de participer au jury. Ses belles mots d'encouragement et de reconnaissance envers mon travail ne pourront que m'inciter à continuer mes recherches sans se lasser ou baisser les bras.

Enfin, je voudrais adresser mes remerciements à Mme. Lucile SASSATELLI, Maitre de Conférence à l'Université Nice Sophia Antipolis - France, pour avoir accepté de rapporter et examiner ce travail et aussi participer au jury de ma soutenance. Le grand intérêt qu'elle a exprimé vis-à-vis mes travaux m'a apporté une grande motivation.

Ces années de thèse ont été remarquablement riches d'expériences et de belles rencontres qui m'ont particulièrement marquées. J'avais l'occasion fréquenter des personnes qui ont été

toutes intéressantes les unes des autres, et qui ont contribué positivement à mon apprentissage, chacun à sa façon. J'aimerais exprimer ma reconnaissance envers tous les membres du laboratoire LabMIA, particulièrement, Mohammed, Charifa et Janati, et également les membres du laboratoire L3I, particulièrement M. Jean Marc Ogier pour son enthousiasme, Mme Nibal Nayef pour son grand soutien, Imen, Maroua, Christophe, Daouda, Nam et Mickael.

Finalement, il y a des personnes que je ne saurais jamais remercier assez. A mes très chers parents qui n'ont jamais cessé de m'encourager et me soutenir pour aller de l'avant, que Dieu le tout puissant leur accorde la santé, la longévité et le bien-être. Je remercie aussi mes petites sœurs et toute ma famille pour leur soutien tout au long de mes études.

Acknowledgment

I am grateful to Allah Almighty for giving me the courage, patience, determination as well as guidance to accomplish this thesis.

I would like to express my special gratitude to Professor El Mamoun SOUIDI for trusting in my abilities, providing me the opportunity of studying at LabMIA and for his unlimited support, advice and encouragements. I would also like to extend my heartfelt gratitude to Professor Arnaud REVEL for providing me the opportunity to work with him. I could have never been able to achieve my goals without his intellectual support and enthusiasm. He has been a source of constant motivation for me and will remain in the future as well.

I wish to extend my utmost gratitude to everyone at LabMIA for providing conducive environment for research and development. I would like also to thank the members of L3I for making my stay wonderful and memorable at La Rochelle.

This thesis in co-tutelle was supported by the grant of the project Hubert Curien Volubilis N° **27041QL**.

I would like to dedicate my life and my work to my beloved parents. I know that I cannot express my gratitude to my parents by words. Throughout my life, they were always there praying for me and for my success, they provided me with every comfort despite of a lot of constraints. Whenever I am feeling depressed and demotivated, they consoled and encouraged me. My parents have always been a source of inspiration, power and motivation for me. They encouraged me at every step of my educational career and made me believe in my abilities. They surrounded me with love, support and consideration. I know I can never thank my parents enough for all the sacrifices they made for me. I pray that may Allah Almighty give them a healthy long life so that they can see their dreams coming true. Ameen.

Abstract

Recently, the distributed computing has witnessed a great evolution due to the use of mobile agent paradigm, endowed with innovative capabilities, instead of the client-server system where the applications are bound to particular nodes in networks. Having captured the interest of researchers and industry, the mobile agents are able to autonomously migrate from one node to another across the network, transferring their code and data, which allows them to efficiently perform computations, gather information and accomplish tasks.

However, despite its significant benefits, this paradigm still suffering from some limitations that obstruct its expansion, primarily in the area of security. According to the current efforts to investigate the security of mobile agents, two categories of threats are considered. The first one concerns the attacks carried out on the mobile agent during its travel or stay by malicious hosts or entities, while the second one deals the attacks performed by a malicious mobile agent in order to affect the hosting platform and consume its resources. Thus, it is substantially needed to conceive a complete security infrastructure for mobile agent systems, which includes methodology, techniques and validation.

The aim of this thesis is to propose approaches which provide this technology with security features, that meet with its overall structure without compromising its mobility, interoperbility and autonomy capabilities. Our first approach was based on XML serialization and cryptographic primitives, in order to ensure a persistent mobility of agent as well as a secure communication with hosting platforms. In the second approach, we have conceived an alternative to the first approach using binary serialization and Identity-based cryptography. Our third approach was proposed to introduce anonymity aspect to the mobile agent, and provide him with a tracing mechanism to detect intrusions along its trip. The fourth approach was developed in order to restrict the access to the resources of the agent platform, using a well-defined access control policy based on threshold cryptography. At this stage, we find it interesting to experiment the utility of mobile agents with security features in preserving the security of other technologies such as cloud computing. Thus, we have developed an innovative cloud architecture using mobile agents endowed with cryptographic traces for intrusion detection and a revocation protocol based on trust threshold for prevention.

Key-Words: Mobile Agents Security, Cryptography, Serialization, Access Control, Intrusion Detection and Prevention, JADE

Résumé

Récemment, l'informatique distribuée a connu une grande évolution en raison de l'utilisation du paradigme des agents mobiles, doté d'innovantes capacités, au lieu du système client-serveur où les applications sont liées à des noeuds particuliers dans les réseaux. Ayant capturé l'intérêt des chercheurs et de l'industrie, les agents mobiles sont capables de migrer de manière autonome d'un noeud à un autre à travers le réseau, en transférant de leur code et leurs données, ce qui leur permet d'effectuer efficacement des calculs, de recueillir des informations et d'accomplir des tâches.

Cependant, en dépit de ses avantages significatifs, ce paradigme souffre encore de certaines limitations qui font obstacle à son expansion, principalement dans le domaine de la sécurité. Selon les efforts actuellement déployés pour évaluer la sécurité des agents mobiles, deux catégories de menaces sont considérées. La première catégorie concerne les attaques menées sur l'agent mobile lors de son voyage à travers des hôtes ou des entités malveillantes, tandis que la seconde catégorie traite les attaques effectuées par un agent mobile illicite afin d'affecter la plate-forme d'hébergement et de consommer ses ressources. Ainsi, il est substantiellement nécessaire de concevoir une infrastructure de sécurité complète pour les systèmes d'agents mobiles, qui comprend la méthodologie, les techniques et la validation.

L'objectif de cette thèse est de proposer des approches qui fournissent cette technologie avec des fonctionnalités de sécurité, qui correspondent à sa structure globale sans compromettre ses capacités de mobilité, l'interopérabilité et l'autonomie. Notre première approche est basée sur la sérialisation XML et des primitives cryptographiques, afin d'assurer une mobilité persistante de l'agent ainsi qu'une communication sécurisée avec les plates-formes d'hébergement. Dans la seconde approche, nous avons conçu une alternative à la première approche en utilisant la sérialisation binaire et la cryptographie à base de l'identité. Notre troisième approche introduit l'aspect d'anonymat à l'agent mobile, et lui fournit un mécanisme de traçage pour détecter les intrusions le long de son voyage. La quatrième approche a été développée dans le but de restreindre l'accès aux ressources de la plate-forme de l'agent, en utilisant une politique de contrôle d'accès bien définie à base la cryptographie à seuil. A ce stade, on s'est intéressé à expérimenter l'utilité des agents mobiles avec des fonctionnalités de sécurité, dans la préservation de la sécurité des autres technologies, telles que le Cloud Computing. Ainsi, nous avons proposé une architecture innovante du Cloud, en utilisant des agents mobiles dotés de traces cryptographiques pour la détection d'intrusion et d'un protocole de révocation à base de seuil de confiance pour la prévention.

Mots-Clés : Sécurité des Agents Mobiles, Cryptographie, Sérialisation, Contrôle d'Accès, Détection et Prévention d'Intrusion, JADE

Résumé Détaillé

Contexte de l'Etude

Actuellement, le besoin en information devient très exigent et l'interopérabilité des systèmes deviens une approche primordiale. La mutation du paysage informatique vers les systèmes distribués, implique l'interaction et la coopération entre les entités et les composants de ces systèmes à travers le réseau, avec des connexions impérativement non-permanentes. Ceci soulève le point de plus en plus sur les aspects de dynamicité, adaptabilité, autonomie et mobilité. Les systèmes multi-agents ont été conçu afin d'apporter l'appui pour ces aspects et résoudre les problèmes qui lui sont adjacents.

L'approche multi-agent [**Fer99**] considère tout système comme une petite société où des entités ou des acteurs autonomes et indépendants appelés "agents" interagissent collectivement en vue de traiter un problème ou exécuter une tâche. Ce qui explique son usage dans diverses disciplines, à savoir la robotique [**Sor13**], l'industrie [**Met11**], l'optimisation des systèmes de transport [**Chen10**], la simulation et la modélisation des systèmes complexes.

Le paradigme d'agent s'est tourné par la suite à la gestion de la mobilité, en inspectant des compatibilités et des convergences assez intéressantes. Les systèmes d'agents mobiles sont une technologie assez innovante qui a pu trouver une place au sein d'applications complexes et émergentes comme le commerce électronique, les télécommunications, la gestion des réseaux distribués [**Gav09**], la recherche adaptative et personnalisée d'information [**Lu07**], les jeux informatiques [**Dig12**], etc. Grâce à leur autonomie, indépendance, adaptabilité et notamment mobilité, les agents mobiles ont la capacité de réaliser leurs objectifs de manière fiable et très flexible. Les agents mobiles se déplacent d'un nœud à un autre à travers le réseau, en possession de toutes les ressources : données, code et état d'exécution, leur permettant d'exécuter leurs tâches indépendamment des hôtes vers lesquels ils migrent.

Pourtant, la mobilité et la flexibilité des agents soulève des problèmes au niveau de la sécurité. Les systèmes à base d'agents mobiles font souvent recours à des interactions locales ou distantes avec d'autres agents sur le réseau. Ces interactions initient des communications difficile d'assurer qu'elles sont saine et sure et ne portent aucun types d'anomalies ou de vulnérabilités susceptibles de nuire à l'agent mobile ou aux hôtes le recevant. Actuellement, ce problème devient un obstacle devant l'expansion de ce paradigme, qui suscite fortement la mise en place de politiques de sécurité, identifiant les spécifications et les besoins à définir pour chacune des entités (agent, hôte). En prenant en considération l'aspect dynamique de ces systèmes qui génère en continue de nouvelles attaques et qui exige que chacune des entités interlocutrices s'adapte au la politique de sécurité de l'autre.

Problématique

L'agent mobile est une approche émergente qui reflète l'autonomie et l'indépendance en prise de décisions et en exécution de tâches. Cet approche rencontre toujours des contraintes à être conçue en pratique à cause de la complexité des principes et des aspects qui lui sont associés. La majorité des travaux en littérature qui traitent les systèmes à base d'agents mobiles, se concentrent sur l'étude de la modélisation à travers différent niveaux d'abstraction : formel, structurel et comportemental, sans se soucier ou accorder une importance aux aspects liés à la sécurité. Ainsi, il semble être nécessaire, voire crucial, d'accorder plus d'intérêt aux problèmes de sécurité soulevés par cette technologie.

En effet lorsqu'un agent se déplace, il est crucial de s'assurer qu'il sera exécuté correctement en toute sécurité sur le nouveau système visité. De même, il est crucial de rassurer le système d'agents de manière qu'il n'y aura aucun risque d'accueillir un nouvel agent mobile. Plus précisément, durant la mobilité de l'agent d'un hôte à un autre à travers le réseau, il peut se trouver face à deux situations qui déclenchent des vulnérabilités compromettant sa sécurité. La première situation s'établit quand l'agent migre vers un hôte malicieux sans pouvoir l'identifier. Ce dernier peut alors exploiter les ressources de l'agent, modifier ses données, profiter de ses comportements et de ses résultats ou encore lui injecter du code malicieux qui infectera par la suite tous les hôtes que l'agent planifie de visiter. ceci reflète éventuellement l'émergence en continu et en chaînage des menaces à d'autres hôtes. La deuxième situation concerne les menaces qui surgissent suite aux contact non sécurisé que peut avoir l'agent mobile avec d'autres agents rencontrés en chemin. Ces agents peuvent avoir des comportements malicieux et bien évidement nuire aux données et aux ressources de l'agent, ce qui lui empêcherait de s'exécuter proprement.

De nombreux efforts ont été consacrés pour enquêter sur la sécurité des systèmes d'agents mobiles. Ainsi, certains aspects fondamentaux tels que l'authentification, la confidentialité, l'intégrité et le contrôle d'accès doivent être adressées. Dans ce contexte, plusieurs investigations ont été mené sur les problèmes de sécurité et les fameuses solutions qu'ont été proposées dans ce domaine, et qui comprennent de nombreuses techniques classiques dédiés à la protection de l'agent mobile et les plates-formes d'hébergement [**Alf05**].

La majorité de ces solutions reposent sur des mécanismes cryptographiques bien connus et des modèles traditionnels de contrôle d'accès [**Bella04**]. A titre d'exemple, le "Traçage d'Exécution" produit des enregistrements des comportements et des actions de l'agent en utilisant les signatures numériques. "L'Encapsulation Partielle du Résultat" préserve les résultats de l'exécution de l'agent en utilisant le cryptage, la signature numérique, la fonction de hachage et le code d'authentification. "L'Obfuscation de Code" impose une transformation au code de l'agent, en utilisant des applications cryptographiques de différents niveaux de confiance. "Code Porteur de Preuve" favorise la vérification automatique du code avant son exécution, sur la base de la conformité d'une preuve codée. "L'État d'évaluation" est basée sur l'utilisation des fonctions d'évaluation, qui déterminent les privilèges accordés à un agent en fonction de facteurs conditionnels et invariants. "Le Bac à Sable (SandBoxing)" permet l'exécution du code de l'agent mobile dans un environnement restreint (bac à sable), qui apparaît semblable au système global, et où les restrictions affectent certaines opérations du code.

En plus de ces solutions, il y a des contributions récemment proposées et qui adoptent de nouvelles visions et orientations du paradigme des agents mobiles. En général, la plupart de ces contributions sont inspirées des techniques précédemment mentionnées ou leur apportent des améliorations, en tenant compte de l'utilisation croissante des systèmes d'agents mobiles dans plusieurs disciplines, avec différents concepts et architectures.

Objectifs de la Thèse

L'objectif principal de cette thèse est la protection des systèmes basés sur les agents mobiles. D'une part, cela concerne la protection de l'agent mobile, y compris son code et ses données, contre les entités malveillantes qu'il peut rencontrer dans son itinéraire, et qui visent à affecter son authentification, sa confidentialité et son intégrité. D'autre part, cela aussi comprend la protection des plates-formes à agents contre les agents mobiles hébergés qui peuvent conduire des comportements malveillants. Par conséquent, l'idée de base est de concevoir des mécanismes de protection qui préservent les exigences de sécurité de ces systèmes, sans pour autant compromettre leurs caractéristiques de flexibilité, d'autonomie et de mobilité.

Dans ce contexte, nous proposons quatre approches :

1. Une robuste authentification basée sur un protocole d'échange de clés, intégré avec la signature numérique. Ce mécanisme est associé à une mobilité fiable de l'agent à l'aide de la sérialisation XML et des primitives cryptographiques [**Idri14a**].
2. Une forte authentification en utilisant le protocole d'accord de clé basé sur l'identité et la signature de Schnorr, ainsi que la combinaison de la sérialisation binaire avec des primitives cryptographiques pour assurer la mobilité sécurisée de l'agent [**Idri15b**].
3. Un processus robuste d'authentification anonyme en utilisant la cryptographie à courbe elliptique avec appariement bilinéaire, et un mécanisme de détection d'intrusion basé sur le traçage d'exécution des comportements et des actions effectués par l'agent mobile sur les plates-formes d'hébergement visitées.
4. Une puissante politique de contrôle d'accès, basée sur un modèle d'accès discrétionnaire associé à un système de partage de seuil, afin d'élaborer une gestion fiable des clés pour les différents droits d'accès à accorder sur les ressources de la plate-forme d'agents [**Idri15c**].

Toutes ces approches sont mises en œuvre et évaluées afin de prouver leur fiabilité, efficacité et sécurité, par rapport aux autres solutions existantes. En outre, une application à la sécurité du cloud computing est proposée, afin de montrer pratiquement que la sécurité de l'agent mobile peut être bénéfique pour d'autres technologies.

Organisation de la Thèse

La présente thèse comprend six chapitres. Les deux premiers chapitres présentent les principaux concepts utilisés, afin de permettre aux lecteurs de se familiariser avec le domaine concerné et sa problématique soulevée. Les trois chapitres qui suivent sont consacrés à la description des approches proposées. Alors que le dernier chapitre présente une application de la sécurité des agents mobiles à la téchnologie du cloud computing.

Le chapitre 1 introduit un état de l'art, où les concepts généraux des systèmes d'agents mobiles sont décrits. Cela inclut leurs définitions habituelles, leurs qualités, leurs domaines d'application, les services nécessaires à leur exécution, les normes adoptées et des exemples de plate-formes pour leur développement.

Le chapitre 2 expose en détail les problèmes de sécurité soulevés par cette technologie. Par la suite, nous énumérons les différentes attaques possibles en fonction de quel aspect de sécurité est affecté, ainsi que les principales contre-mesures qui ont été proposées et examinées par les chercheurs.

Dans le chapitre 3, deux approches basées sur le mécanisme de sérialisation sont décrits. La première approche utilise la sérialisation XML afin d'assurer un format persistant pour la mobilité des agents, tandis qu'un protocole d'échange de clés intégré avec la signature numérique est utilisé pour élaborer un processus d'authentification entre l'agent et sa plate-forme d'hébergement [**Idri15a**]. Dans la seconde approche, une sérialisation binaire est adoptée pour permettre une migration souple de l'agent, alors qu'une authentification forte est proposée, basé sur un protocole d'accord de clé qui utilise l'identité et la Schnorr signature [**Idri15b**].

Dans le chapitre 4, on propose un système d'authentification qui préserve l'anonymat de l'agent mobile lorsqu'il est hébergé par des plates-formes d'exécution. Ce système est basé sur la cryptographie à courbe elliptique et l'appariement bilinéaire. En outre, un procédé de détection d'intrusion est proposé, en faisant usage de la technique des traces cryptographiques, où les comportements et les actions effectués par l'agent mobile sur chaque plate-forme d'hébergement visitée sont enregistrés jusqu'à ce que l'agent retourne à sa plate-forme propriétaire, où ils peuvent être vérifiés.

Le chapitre 5 propose une politique de sécurité pour protéger la plate-forme d'hébergement et ses ressources. Cette politique utilise le modèle de contrôle d'accès discrétionnaire (DAC) pour concevoir une matrice d'accès aux ressources de la plate-forme. En outre, un schéma de partage de secret à seuil est intégré afin de dériver un ensemble de clés d'accès à partir de la clé principale de la plate-forme, et qui sont nécessaires afin d'attribuer les droits d'accès sur les ressources associées aux légitimes utilisateurs [**Idri15c**].

Dans le chapitre 6, nous décrivons une nouvelle application de la sécurité des agents mobiles pour assurer une communication sécurisée entre les entités du cloud computing. A cet effet, une détection d'intrusion et de prévention pour le cloud computing (cloud-IDPS) est conçue, en utilisant deux techniques principales : une exécution de traçage pour garder une preuve sur les services et les calculs effectués sur les serveurs du cloud, ainsi qu'un protocole de révocation basé sur un seuil de confiance afin d'empêcher les intrusions détectées de se propager [**Idri15d**].

Par la fin, cette mémoire se termine par une conclusion générale qui résume nos contributions et décrit les futures perspectives.

Contents

List of Figures

List of Tables

General Introduction

Contents

Context

Recently, the need for availability of information is exigent, and the interoperability in IT systems becomes of a feature of a paramount interest. This is due to the mutation of the IT landscape into distributed systems, which implies the interaction and cooperation among all entities and components of that system across the network, with connections that are imperatively non-permanent. Accordingly, many essential aspects are increasingly highlighted, such as: autonomy, dynamicity, adaptability and mobility. Among the newly emerging technologies in the field, multi-agent systems (MAS) have been designed to provide support for these aspects and solve problems that are adjacent to them.

According to Ferber [**Fer99**], the multi-agent approach considers every system as a miniaturized society, where autonomous and independent entities, called "agents", interact collectively to address a problem or perform a task. This conception explains its increasingly usage in a wide variety of disciplines, namely in robotic [**Sor13**], industry [**Met11**], the optimization of transport systems [**Chen10**], the simulation and modeling of complex systems.

For more reliability and effectiveness, the agent paradigm is eventually turned into the mobility management, through inspecting more interesting capabilities and convergences. The mobile agent systems are viewed as an innovative technology, that has managed to get a place, within complex and emerging applications, such as: electronic commerce [**Fas07**], management and optimization of distributed networks (especially wireless sensor networks) [**Gav09**], adaptive and personalized information retrieval [**Lu07**], computer games [**Dig12**] and many others. Through their autonomy, independence, adaptability and mobility in particular, mobile agents have the ability to achieve their goals, in flexible and operational manner. Indeed, the mobile agents move from one node to another over the network, in possession of all needed resources: data, code and execution state, which enable them to perform their tasks independently of the hosts to which they migrate.

However, mobility and flexibility of agents raise many problems at the security level. This is due to the fact that systems based on mobile agents often make use of local or remote interactions with other agents on the network. These interactions initiate to a communication,

whose it is difficult to ensure that it is safe and secure, and do not carry any anomalies or vulnerabilities that could affect the mobile agent or its receiving host. Thus, security problem becomes an obstacle to the expansion of this paradigm, which greatly promotes the implementation of security mechanisms and policies, that identifies requirements and specifications to be defined for each entity (agent, host). This cannot be achieved without taking into consideration, the dynamic aspect of these systems, that are continuously victims to new attacks, and which require each of the interlocutor entities to be adapted with the security policy of the other.

Problematic and Related Works

Mobile agent is an emerging approach, that reflects the autonomy and independence in decision-making and execution of tasks. Yet, this approach still faces constraints to be designed in practice, because of the complexity of the principles and aspects associated with it. The majority of literature works, dealing the systems based on mobile agents, focus on the study of modeling through different abstraction levels: formal, structural and behavioral, without caring or attach importance to the aspects related to security. Hence, it seems to be necessary, even crucial, to grant more interest in security issues raised by this technology.

Indeed, when an agent is moving, it is crucial to ensure that it will properly and safely executed on the new system visited. Similarly, it is crucial to reassure the agent system, that there will be no risk to welcome a new mobile agent. During the mobility of the agent from one host to another over the network, it may face two situations that trigger a set of vulnerabilities compromising safety. The first situation occurs when the agent migrates to a malicious host without being able to identify him. In this case, the malicious host can exploit the resources of the agent, modify its data, take advantage from its behaviors and results, or even more, inject malicious code that will subsequently infect all the intended hosts in the itinerary of the agent. Thereby, this eventually reflects the continuous and chained emergence of the threats to other hosts. The second situation involves the threats, that arise when the mobile agent gets untrustworthy contacts with other agents encountered along its way. These agents may adopt malicious behaviors, that could obviously prevent the mobile agent from being properly executed, by harming its data and resources.

Many efforts have been devoted to investigate the issue of mobile agent systems security. Thus, in terms of security, some basic issues such as authentication, confidentiality, integrity and access control should be addressed. Towards this, Alfalayleh et Brankovic [**Alf05**] conducted a survey on the security issues and basic solutions in that field, which include many conventional techniques dedicated to protect the mobile agent and the hosting platforms.

The majority of these solutions rely on well-known cryptographic mechanisms and traditional access control models [**Bella04**]. For instance, "Execution Tracing" produces records of the agent's behaviors and actions using digital signatures. "Partial Result Encapsulation" preserves the results of the agent's execution using encryption, digital signature, hash function and authentication code. "Code obfuscation" enforces a transformation to the agent's code, using cryptographic maps with various trust levels. "Proof Carrying Code" promotes the automatic verification of the code before its running, based on a conformity of an encoded evidence or proof. "State Appraisal" is based on the use of appraisal functions, which determine the privileges granted to an agent based on conditional factors and invariants. "Sandboxing" allows the execution of the mobile agent's code in a restricted environment (sandbox), that appears similar to the global system, and where restriction affects certain code operations.

In addition to these solutions, there are recently proposed security contributions that adopt

new visions and directions of the mobile agent paradigm. In general, most of these contributions are inspired from the techniques previously mentioned or bring enhancements to them, taking into consideration the increasing use of mobile agent systems in multiple disciplines, with different conceptions and architectures.

Zhang et al [**Zhan06**] describes how remote policy enforcement can be used for runtime protection of agent's code. His paper provides good mechanisms for restricting unauthorized users from executing the code, which in turn provides good security for code of mobile agents. They have shown how a Java Authentication and Authorization Service (JAAS) can be used to enforce access control restrictions over data. However, the downfall is that system uses "Trusted Computing Devices" and depends on "Trusted Runtime Environment (TRE)". Therefore, in order this system to work, TRE should be present.

Mubarak et al [**Mub07**] describes semantically rich security policies for mobile agents. According to the authors, policy enforcement consists of policy storage area, policy distributor, policy implementer, and monitor and policy specifier. They structured policies for mobile agents into three different types: authorization policies, obligation policies, and refrain policies. These policies address issues associated with agents' security, like encryption, access control, and authorization. Although they have categorized policies, they did not specify the structure of a policy.

Shibli et al [**Shi10**] presented a solution for authentication and authorization of mobile agents based on Role-Based Access Control (RBAC) and eXtensible Access Control Markup Language (XACML) policies. This solution can be applied in two steps: a) by creating security architecture for authorization of mobile agents (authorization system), and b) by specification of the structure/model for role-based XACML policies for mobile agents. These two steps combined provide a complete authorization system for mobile agents.

Ibharalu et al [**Ibh11**] have proposed the use of a chain of digital envelopes with platform registries to support dynamic agent's itineraries in open network environment. This scheme protects and allows mobile agents to roam freely in open networks environment without being compromised in a malicious hosts. The main advantage is that the proposed scheme exhibited better performance when compared to the results obtained from obfuscation methods in terms of data integrity and security. The main drawback is that the proposed scheme consumes a little more time visiting platform registries and executing complex cryptographic functions than the obfuscation methods. Though the data is protected from hosts, the code is vulnerable to attack by other malicious agents residing in the host.

Menacer et al [**Men11**] have presented a new mobile agents-based architecture that intends to provide a comprehensive solution to the various issues related to security. This architecture consists of a generalization of the market mechanisms to the non-market systems and it relies on an extended mobile agent model, the seller-buyer (SB) model. Thus, it provides developers with the opportunity to safely and efficiently use mobile agents in order to build distributed large-scale applications.

Razouki and Hair [**Raz14**] have introduced an approach to find a new mobile agent paradigm architecture, which can protect the mobile agent via two strategies of adaptation. The first strategy a static adaptation performed by the MSAS (Management System of Agents Security) based on the sensitivity of the services requested by the agent. The second strategy consists of a reflexive dynamic structural adaptation performed by the mobile agent itself. According to the degree of confidence on the platform visited, the mobile agent selects and adapts the security components to the tasks to be performed by this platform.

Srivastava and Nandi [**Sri14**] have proposed a security protocol built on the foundation of integrity based confidentiality and self-protection approach based on agent driven security.

Thus, the self-protection makes the mobile agent less interactive during its execution, while the agent driven security is achieved through a novel concept of symmetric key's component distribution. According to this latter, a key component is distributed in secure manner, and another key component is derived from ensuring integrity of data collected at run time. The validity of this approach in overcoming different kind of security attacks, Petri net based formal representation of the security protocol is provided to strengthen the belief of distributed applications.

Objectives

The primary objective of this thesis is the protection of the mobile agent systems. This concerns, in one hand, the protection of the mobile agent including its code and data, against malicious entities it may meet in its itinerary, and which aim at affecting its authentication, confidentiality and integrity. On the other hand, this implies the protection of the agent's platforms against the hosted agents which may conduct malicious behaviors. Then, the basic idea is to design security mechanisms that preserve the privacy requirements of these systems, without compromising their flexibility, autonomy and mobility features.

In this context, we have proposed four approaches:

1. A robust authentication process based on a key exchange protocol integrated with digital signature, as well as a reliable mobility of the agent using the XML serialization and the cryptographic primitives [**Idri14a**].
2. A robust authentication process using ID-based key agreement protocol along with Schnorr signature, as well as a binary serialization combined with cryptographic primitives to ensure secure mobility of the agent [**Idri15b**].
3. A robust anonymous authentication using elliptic curve cryptography along with bilinear pairing, and a detection intrusion mechanism based on the execution tracing of the behaviors and actions performed by the mobile agent on the hosting platforms.
4. A strong access control policy based on a discretionary access model associated to a threshold sharing scheme, in order to elaborate a keys management for the different access rights to be granted on the platform resources [**Idri15c**].

All these approaches are implemented and evaluated to prove their reliability, efficiency and security, compared to other existing solutions. In addition, an application to cloud computing security is proposed, in order to prove how much the security of mobile agents can be beneficial for other technologies.

Organization of the Thesis

The present thesis is organized into six chapters. In order to enable readers to be familiar with the concerned field and its problematic attested, the first two chapters introduce the main concepts to be used and discussed. The following three chapters are devoted to the description of the proposed approaches. while the last chapter presents an application of mobile agent security to the cloud computing technology.

Chapter 1 introduces a state of the art, where the general concepts of the mobile agent systems are described. This includes their usual definitions, their qualities, their application areas, the services required for their execution, the adopted standards and platform examples for their development.

Chapter 2 exposes in details the security problematic in this technology. Subsequently, we enumerate the different possible attacks according to which security aspect is affected, and the principal counter-measures that have been proposed and considered by researchers.

In Chapter 3, two approaches based on the serialization mechanism are described. The first one makes use of the XML serialization to ensure persistent format for the agent mobility, while a key exchange protocol integrated with digital signature is used to elaborate an authentication process between the agent and its hosting platform [**Idri14a**]. In the second approach, a binary serialization is adopted to allow a flexible migration of the agent, while a strong authentication is proposed, based on ID-based key agreement protocol along with Schnorr signature [**Idri15b**].

In Chapter 4, we propose an authentication scheme that preserves the anonymity of the mobile agent when being hosted by the executing platforms. This scheme is based on the elliptic curve cryptography and the bilinear pairing. In addition, a detection intrusion method is proposed, which is based on cryptographic traces technique. In this latter, the behaviors and actions performed by the related agent on each hosting platform are recorded until the agent returns back to its owner platform, where they can be verified in case of possible suspicious entities.

Chapter 5 proposes a security policy to protect the hosting platform and its resources. This policy makes use of the discretionary access control model (DAC) to design an access matrix to the platform resources. Besides, a threshold sharing scheme is integrated to derive a set of access keys from the platform master key, and which are required to be granted the access rights of the associated resources [**Idri15c**].

In Chapter 6, we describe a new application of the mobile agent security to secure the communication among the entities of the cloud computing. For that purpose, an intrusion detection and prevention system for cloud computing (cloud-IDPS) is designed, making use of two principal techniques: an execution tracing to save a proof of the services and computations performed on the cloud servers, as well as a revocation-based trust threshold protocol in order to prevent the intrusions detected from spreading [**Idri15d**].

Finally, this memory is ended with a general conclusion that summarizes our contributions and describes some perspectives.

Chapter 1

Mobile Agent Systems

Contents

Nowadays, the development of large-scale networks has enabled the emergence of a wide variety of new technologies, such as: electronic commerce, information retrieval, system optimization, distributed computing and many others. In these technologies, the communication among their entities is very substantial to ensure their proper functioning. In this context, the concept of mobile agents appeared as an innovative solution, that enables the implementation of dynamic and adaptable applications, and makes the development of distributed applications on large networks more generic.

A mobile agent is an autonomous entity or program running in an environment, using its own resources and a set of services provided by the hosting platform. In this chapter, we will define

the main notions and services provided by the mobile agents and their associated environments. We begin by exposing the evolution of the communication and exchanges in networks, starting from client/server till mobile agent paradigm. We will process the life cycle of the mobile agent, and discuss the qualities and advantages of this technology, including the mobility feature. The standardization efforts are also evoked to highlight the interoperability, and the compatibility among the environments and platforms executing the mobile agents. Examples of commonly used platforms are defined and compared, according to many metrics, in order to show the most suitable one for the modeling and implementation of our contributions.

1.1 Evolution

1.1.1 Client/Server Architecture

The client/server architecture is a model of software running, that can be realized between inter-connected physical structures. It is based on two types of software or application: client software and server software, that are running on two different machines connected through communication channels. Within this architecture, an interaction between a client and server is performed as illustrated in Figure 1.1. Accordingly, a client machine contacts a server to provide services or carry out treatments. These requested services, viewed as programs acting on data, are exploited by clients on remote machines, that are able to process the information they retrieve from the server.

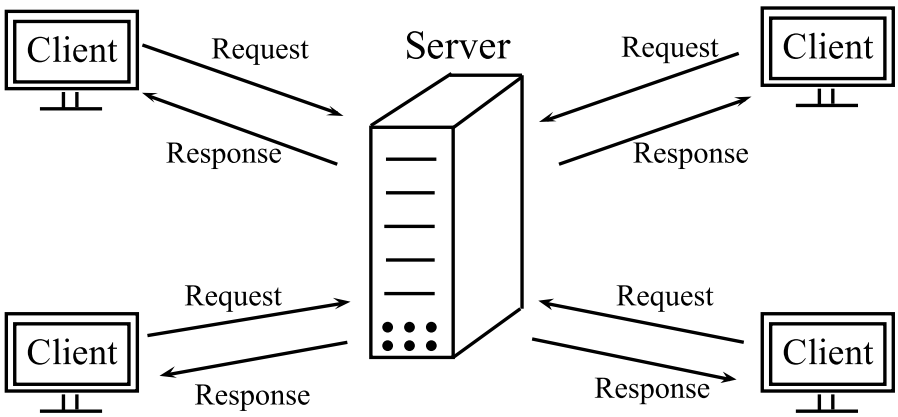

Figure 1.1 – Client/Server Architecture

In general, there is no standard definition for a client/server architecture. However, there are some major concepts that govern around:

— *Protocol*: it is a standard that assembles rules and procedures, to ensure communication between processes running on different machines, transmitting and receiving data over the network. It is always the client who initiates the communication via services requests, while the server waits passively to be activated and revived.

— *Heterogeneity*: It is a concept clearly shown in the different nature of hardware and software structures (operating system or file management system), as well for one or multiple networks. In other words, this implies the independence of the client/server

architecture from the platforms where it operates.
— *Flexibility and Adaptability*: The client/server architecture has great flexibility, insofar as the server module can be adjusted by an interface, an application, a station or a newer and upgraded model, without affecting the client module. Also, the reverse is possible.
— *Resize*: The hardware structure of the client/server model can be expanded or reduced by adding or removing client stations. This is also valid for the software structure, where the server can be upgraded with applications and data, that are newer and more sophisticated.

Client/Server Models

There are different models of the client/server architecture, depending on the treatments and services provided by the server.

— *Processing Model*: in this model, the client requests the server to perform treatments on his behalf, and provide the corresponding results. These treatments may include operations on data, input and verification of forms, alarm processing, etc. In addition, they are performed using programs implemented on servers, or incorporated into databases.
— *Presentation Model*: in this model, the presentation of interface displayed to the client is managed by the server. In this context, we distinguish two types. The "remote presentation", where the client only supports the display of the interface, which greatly increases network traffic and prevents load sharing between client and server. The "distributed presentation" corresponds to a framing of the display in a character mode, under the direction of a central site, but this makes the model more abusive, knowing that the client always retains the slave role regarding the server.
— *Data Model*: it is the most famous model, and it is widely used by the SGBD systems, that are installed at the server machine to ensure the management, storage and processing of data. Thus, the client sends data processing requests as SQL queries to the server, which in turns executes these tasks and sends back the corresponding results.

Advantages and Drawbacks

The client/server architecture is particularly recommended for networks requiring a high level of reliability. This is due to its main features described bellow:

— *Centralized resources*: since the server is placed at the core of the network, it can manage the resources commonly used by the clients (such as a centralized database), in order to avoid problems of redundancy and contradiction.
— *Better security*: because the number of entry points for access to data is less important.
— *Administration at the server level*: the clients with little importance in this model, have less need to be administered.
— *scalable network*: thanks to this architecture, it becomes possible to remove or add clients, without disrupting the running of network or perform major modifications.

However, the client/server still suffers from some flaws, including:

— *High cost*: due to the technical nature of the server.
— *Weak link*: the server is the only weak link in the client/server network, given that the entire network is built around it. Fortunately, the server has a high fault tolerance (notably through the redundant array of independent disks (RAID) system).

1.1.2 Remote Evaluation

The remote evaluation is a technology that belongs to the field of code mobility, where the client and the evaluation site are separated in time and/or space during the processing. An interaction by remote evaluation, as shown in Figure 1.2, occurs when a client sends its code to a distant site to be executed. The receiver site uses its own resources to run the code, and then, deliver the results to the client in an additional interaction. In this scheme, only the code is transmitted to the server, and it is only launched on that server. Usage examples of this technology are: the code of a SQL query issued to a database server, and the interactions among the PostScript printers.

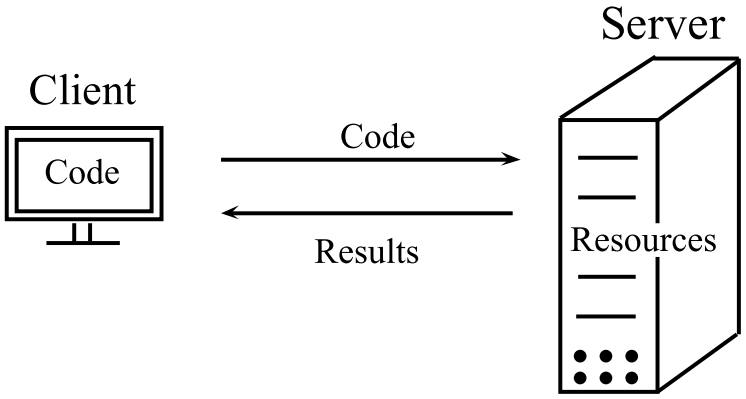

Figure 1.2 – Remote Evaluation Architecture

Advantages and Drawbacks

The remote evaluation technology is becoming increasingly important for the number of advantages it offers, such as:

— The great ability to gather detailed information on the behavior of the actual user in real contexts of use. This is strongly requested and useful in contexts, where it is difficult (or convenient) to install an evaluator to directly observing or recording the session.

— The centralization that allows the clients to carry out the evaluation in their familiar environments, which contributes to ensure more security and gain more through the natural behavior of the users.

Nevertheless, remote evaluation still presents some limitations, such as:

— It becomes very difficult to gather data and track the user's interactions and behaviors, when it concerns applications with limited capabilities (such as mobile devices), that imposes constraints on the kinds of techniques to be used.

— Detecting the environmental conditions in which the session takes place is also an other persistent problem.

1.1.3 Code on-demand

In the code on-demand (COD) technique as illustrated in Figure 1.3, the client has access to a set of resources, but does not hold the skills to process them, then, its interacts with a remote

site in order to gather a knowledge that will be executed on the client machine. Thus, the client charges the necessary code to the achievement of a service, while the remote site is responsible for providing the required service's know-how. The popular application of this technique is the Java Applets, which consist of programs able to be loaded from web pages and executed on the client machine [**Fal06**].

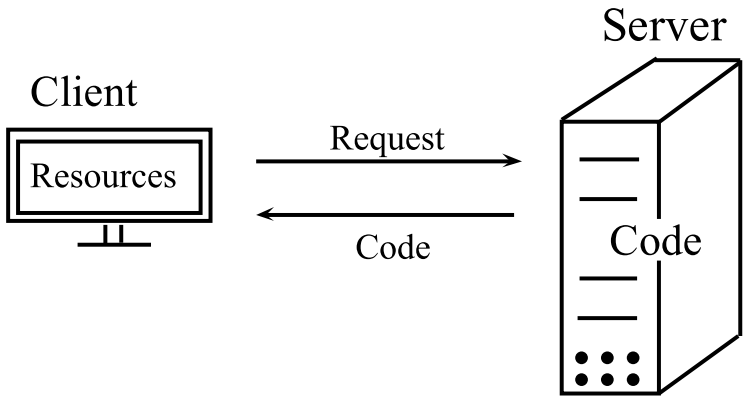

Figure 1.3 – Code on-Demand Architecture

Advantages and Drawbacks

The code on-demand is widely used in web services, due to its significant advantages. Among these advantages, we can cite:

— It allows the client's functionalities to be extended by downloading and executing code in the form of applets or scripts.
— It reduces the number of features required to be pre-implemented.
— It improves system extensibility through allowing features to be downloaded after deployment.
— It ensures that the application will be simple to maintain and easily upgradeable to new technology.

However, the principal drawback of the COD lies in the fact that it reduces of visibility.

1.2 Basic Concepts of Mobile Agents

1.2.1 Fundamentals

Before approaching the concept of mobile agent, we find it interesting to define some important notions, such as: agent and multi-agent system.

Agent

In fact, there is no standard definition for the term of "Agent", since there is a multitude of similar definitions of agents, but differ according to the type of application for which the agent is designed. According to [**Brio01**], the following types of agents are identified:

— *Material Agent*: it is an agent with a physical device, such as a robot.
— *Software Agent*: a software agent is purely logical, which includes the code, data, and status. An example is the Unix daemons.
— *Reactive Agent*: this agent is marked by its simplicity, its small size and its behavior based on the stimulus-response principle.
— *Cognitive Agent*: also called intentional, it is an agent that not only acts according to the conditions of its environment but according to its own goals and intentions.
— *Adaptable Agent*: an agent that some of its internal, operational or functional (behavioral) processes are modifiable running.
— *Social Agent*: it operates advanced interactions and cooperation with other agents in its environment.
— *Mobile Agent*: it is an agent able to move in the environment or migrate from one site to another following an itinerary, in order to be executed according to its tasks.

Among the numerous definitions provided, the majority seem to be a consensus within the multi-agent community. In this context, we adopt the definition of Ferber [**Fer99**], where an agent is viewed as a physical or virtual entity, which:

— has its own resources.
— is able to interact within an environment.
— is able to perceive its environment (but to a limited extent).
— is powered by a set of patterns (individual objectives and satisfaction/survival functions) that seeks to optimize.
— can communicate directly with other agents.
— provides services and owns capabilities.
— may eventually breed.

This definition raises four essential properties:

— *Autonomy*: since an agent is directed by a set of its own trends, which are not only behavioral but also concern its own resources (Memory, CPU, energy, access to sources).
— *Independence*: agents are independent from the environments where they are implanted, which is automatically derived from the autonomy, due to the facts of transporting their own resources and being able to manage them.
— *Flexibility*: it is evaluated when the agent is:
  — *Proactive*: in addition to responding to its environment, the agent is capable to opt an opportunistic behavior, led by its goals or its utility functions, and thus take initiatives when appropriate.
  — *Social*: it is able to interact with other agents (artificial or human) to complete tasks or help others complete theirs.
  — *Able to respond in time*: it can perceive its environment and respond quickly to changes occurring there.
— *Situatedness*: the agent can receive sensible inputs from its environment, and perform actions that are likely to change this environment. The Internet is an example of environments where agents can be situated.

Multi-Agent System

A multi-agent system (MAS) is a system in which a set of intelligent entities called "Agents" are cooperating and coordinating their goals and action plans, to solve a problem or achieve an objective. According to Ferber [**Fer99**], it is a realization of electronic and computer models, consisting of artificial entities that communicate with each other and act in an environment.

Indeed, this is another manner to involve the collective intelligence, while distributing activities on contributors to achieve a goal, knowing that each contributor has a partial view of what to do. A multi-agent system is composed of the following elements, as shown in Figure 1.4:

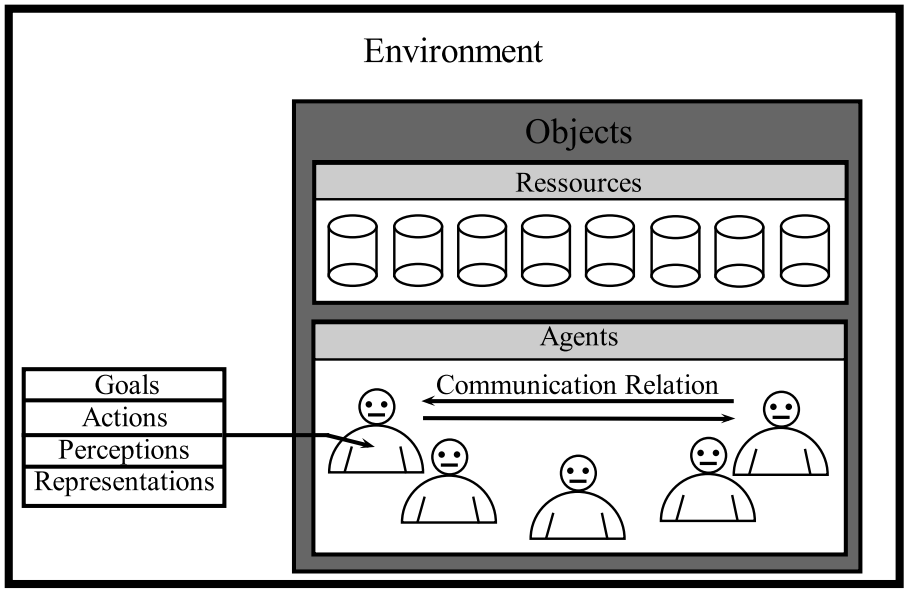

Figure 1.4 – Structure of a Multi-Agent System

— **E**: an environment with a metric.
— **O**: a set of objects. These objects are located, such that each object, at a given time, can be associated to a position in E. They are also passive, that is to say, they can be viewed, created, modified and destroyed by the agents.
— **A**: a set of agents, that are particular active objects ($A \subset O$) of the system.
— **R**: a set of relations that link the objects (Agents) together.
— **Op**: a set of operators, allowing the agents to collect, produce, consume, transform and manipulate objects from O.
— A set of operators responsible for representing the application of "Op", and the reaction of the world face to any attempt to change. It might be called "the laws of the universe."

Mobile Agent

The diversified usage of the mobile agent concept in several disciplines, has resulted in a variety of definitions. Among these definitions, a common and general one is provided as the following:

A mobile agent is an agent that possesses autonomous behaviors, mainly related to its knowledge, its observations and interactions within the environment and with other agents. It is characterized by its ability to move from one site to another in the network, with its set of resources, to accomplish the client's task.

Figure 1.5 shows an illustration of a mobile agent as a physical or software entity able to move from one site to another across the network. During this trip, the agent executes the tasks specified by its owner on the visited sites, while the results are stored until it returns back

to the original platform. Indeed, the mobility aspect is associated to the agent technology, in order to make the latter more appropriate to the needs of large-scale networks and for mobile computing. Thus, it is particularly intended for the implementation of applications, whose performance varies depending on the availability and quality of services and resources, as well as on the volume of data exchanged. According to what is mentioned before, the following essential keys are derived:

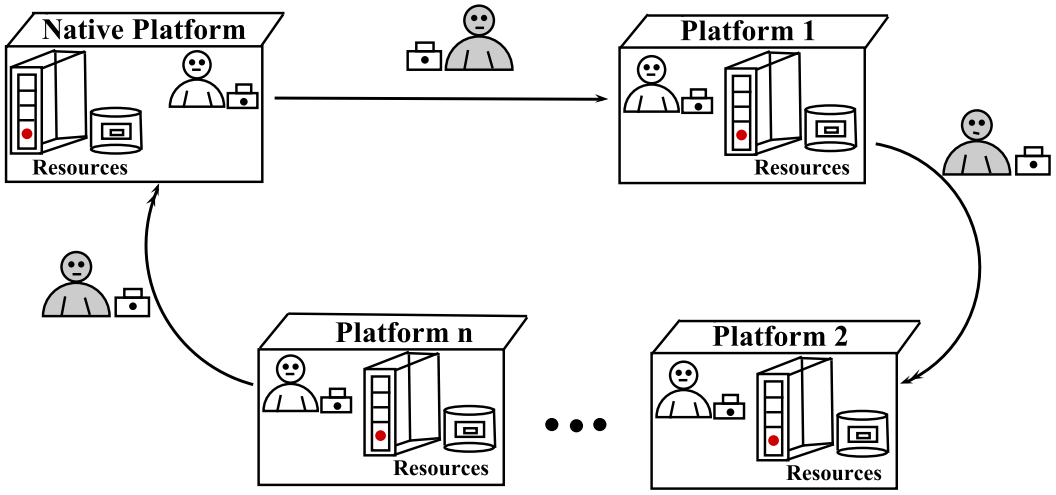

Figure 1.5 – Mobile Agent Concept

- **Environment**: The environment in which the agent is evolving must comprise:
  - Machines that receive the agent while its execution, and which provide the agent with the necessary resources (computing unit, memory, etc.) to achieve the tasks required by the clients.
  - Other stationary and mobile agents, that cooperate with the mobile agent to reach objectives.
  - Network links that facilitates the mobility of the agent across different sites.
- **Memory**: every mobile agent is endowed with a storage capacity, enabling it to gather information on the visited sites. The collected information are stored until return of the agent to the client, to whom they are supplied.
- **Behaviors**: from a conceptual perspective, the agent is provided with a program that threaded the tasks to accomplish. It is called to be reactions by delegation for the client. Thus, the agent may migrate to a hosting platform, with a set of specific competences.
- **Autonomy**: an agent acts independently of its owner and the visited sites. It is able to decide where it will move and what to do there, according to the tasks assigned to it. That is to say, the agent itinerary can not always be predict.

1.2.2 Qualities and Advantages

In literature, there are two type of agents. Stationary agent that resides on a host, and communicates with its environment using conventional techniques, such ad the remote procedure calls (RPC) or the notification messages. However, when they need to interact with other agents

on remote machines, they are obliged to use communication protocols based on client/server model.

Thus, once mobility is associated to agents, it is considered as an orthogonal property. It allows agents to communicate with other agents, and roam freely, while optimizing their itineraries and deciding on the tasks to be executed. This can be achievable even when agents are outside their environments, or these latter are out of service.

As a matter of fact, there are several qualities and advantages that motivate and promote the use of mobile agents technology over the traditional client/server model, particularly in the building of distributed systems. As such, we mention seven characteristics, according [**Lan99**]:

- ***Reduce of the network load***: the communication within the distributed systems usually require multiple interactions, to accomplish tasks through direct transmission protocols, which increases the network traffic. Thus, mobile agents allow clients to package the conversations and dispatch them to the destination host, in order to perform and deal with interactions locally. This becomes more perceptible and useful when reducing the flow of raw data in the network, or when it concerns very big data stored at remote hosts, and which need to be processed in its locality rather than transferred over the network.
- ***Overcome of the network latency***: distributed systems that are processing on real-time mode, like cooperative industrial robots, need to support all changes in their environments and respond to them in real-time. The fact of monitoring such systems via a substantial volume network involves significant latencies, which is unacceptable for sensible systems. For critical real-time systems, such latencies are not acceptable. Making use of mobile agents solves this concern, since these entities are able to migrate to a machine and act locally, in order to execute the controller's directions directly.
- ***Encapsulation of protocols***: during the exchange of data within a distributed system, it is strongly needed that each host holds a mechanism to properly code the outgoing data, and interpret the incoming data. However, since that the communication protocols are evolving to satisfy the new requirements for efficiency or security, it becomes more complicated if not impossible to proceed to this upgrade properly. As a result, protocols often become a legacy problem. In the case of mobile agents, the migration to remote platforms allows the incorporation of "channels" based on proprietary protocols. With this context, the mobile agent is able to access the messages it carries, even if the format of these messages is changing, because the consuming way is also changing.
- ***Asynchronous and autonomous execution***: the proper implantation of mobile devices require expensive network connections, which are continuously open without interruption. This is greatly onerous at economical and technical levels, and cannot be feasible. As a solution for this issue, tasks are eventually involved into mobile agents, which can then be dispatched into the network. At this moment, the agents become independent of their owners and can carry out executions asynchronously and autonomously. Later, the mobile device can reconnect and gather results from the agent.
- ***Dynamic adaptation***: among the qualities of mobile agents is that they are able to sense their execution environment, and thus respond autonomously to changes. Besides, agents may work in communities and spread themselves among multiple hosts in the network, in order to maintain the optimal configuration for solving a particular problem. Such a collaboration is defined, when an agent on host X dispatches another agent to a host Y where no agent is detected, also when a host shuts down, the mobile agents are informed to update their itineraries and eliminate the host in question.
- ***Natural heterogeneity***: from hardware and software perspectives, the network com-

puting is fundamentally heterogeneous. This is mainly due to the optimal conditions provided by mobile agents for seamless systems integration. Normally, these systems can be conceived, basing on the independence of computer-layer and transport-layer, which is achievable in case of mobile agent systems, since these two layers only depend to the execution environment.

— ***Fault-Tolerance and robustness***: it becomes more easier to build fault-tolerant and robust systems through the use of mobile agents, that react autonomously and dynamically to inappropriate events and incidents. For instance, when a host is being shut down, all agents executing on that host are warned to move to another destination, where they can pursue their execution. It is worth to mention that the perimeter of a community in general evolves over time, depending on system failures or the integration of new resources.

1.2.3 Application Areas

In this section, we will discuss the use of systems based on mobile agents in distributed applications. As such, there are three categories of applications:

— **Computing**: where the agent takes advantage of the resources of the visited platform to perform its calculations, which allows the distribution of an execution and its adaptation to the loads of machines. A highly recognized application of the use of mobile agent in this category is:

— *Distributed Computing* [**Cao12**]: the distributed computing aims that every resource has the same workload, which prevents a network node to be inactive, while other tasks are waiting for execution on other nodes. Mobile agents are introduced in this way, to ensure the dynamicity of tasks so that each task is represented by an agent, whose life cycle depends on the duration of the task. This approach benefits from cooperation and autonomy of mobile agents to make decisions of localization tasks.

— **Delegation**: where the agent is charged by a client to complete a task. Among the applications we include in this category are:

— *Electronic Commerce* [**Che14**]: It concerns transactions with an electronic transmission medium (www), and for which, a high heterogeneity is required in order to access them at any time, place and on any device (computer, mobile phone, tablet, TV interactive ...). The introduction of mobile agents in this application allows to gather data, and adapt them according to the sites to which they are intended. This is also assessed for mobile commerce (M-commerce).

— *Finding Information* [**Og15**]: looking for information on the network requires multiple interactions, which ends up with unnecessary information for the client. The use of mobile agents aims to reduce the rate of unnecessary intermediate exchanges. Thus, when an agent is sent to a host, the search and the exchange of messages become local, which releases the network load. In addition, the agent carries only the useful results of research conducted on several hosts.

— **Filtering**: where the agent processes the gathered data, filters and adapts them to the client needs. Among the applications [**Li15**] we include in this category:

— *Network Administration*: it is necessary to ensure optimal use of the network and its services. It involves mechanisms for gathering and visualizing data, detection and management of faults, configuration of actor's behavior and security management. To meet these constraints and many others, the administrative tasks are then assigned to mobile agents to perform their execution independently and flexibly. In

addition to reducing the cost of communication, this technology allows to distribute computations and analyses on several administered sites, which reduces the load on the administration platform.

— *Management of Active Networks*: active networks is a new paradigm, in which blocs of programs are extracted and downloaded, in order to be executed on network nodes to change their behaviors. However, this obviously affects protocols, services and applications, as well as mechanisms and high-level processes. To address this issue, we have introduced mobile agents that encapsulate active programs to be executed, instead of using passive packets of the current network. These mobile agents are transmitted in packets and executed on each visited node by these packets.

1.3 Services for Mobile Agent Execution

1.3.1 Structure, Creation and Communication

Structure of a mobile agent

A mobile agent is an entity or a program able to independently and autonomously execute jobs on behalf of a user. Thus, an agent needs to be provided with the necessary capabilities to migrate from one machine to another across the network, which makes it important to recognize the elements to be joined and transferred with the agent during the mobility process. According to [**Tanen07**], there are three components involved in a mobile agent:

— The Code: a sequence of instructions to be executed, which define the static behavior of the mobile agent.

— The Execution Context: it reflects the current execution state of the mobile agent (registers values, execution stack).

— The Data and Resources: there are two types:

  — Transferable Resources: they represent the attributes values of the agent, which provide him with a global state.

  — Non-Transferable Resources: they constitute the execution environment provided by the system (open files, sockets, connections, registers, etc.) and the physical materials used by the agent (printer, monitor, etc.) .

Creation of a mobile agent

The mobility of agents usually raises the issue of their creation and the locations where they can start their activities. Indeed, the activity of an agent is related to the execution launching of its code, which may have different forms:

— Local creation

— Creation with code on demand

— Creation with remote execution

It is worth to mention that the agent creation uses mechanisms on which rests mobility. In addition, each created agent is provided with a globally unique name, that allows to locate the agent and contact it. This is commonly known by "Naming Service". In the majority of mobile agent systems, this unique name includes the characteristics of the machine (name, IP address, MAC...) where the agent is executed, the port number, the name of the network domain and a locally unique identifier [**Fal06**].

Communication between agents

Generally speaking, the communication between two entities could be performed in two different manners:

1. it is the most intuitive, and makes use of mechanisms allowing direct communication between objects. This corresponds to the synchronous and asynchronous communications.
2. it consists in adopting mechanisms for indirect communication, such that an object willing to communicate with another one sends a message to a third party, that takes the responsibility to forward it to the recipient.

Indeed, it is important in a mobile agent system to recognize if the agent communicates with another agent or with a set of agents. Besides, the agents can communicate with other ones resident in the same place or with agents located in other places. An agent can evoke a method of another agent and send him messages, if it is authorized to do, in a predefined language such as the Agent Communication Language (ACL), which provides the agents with the necessary tools to exchange messages and knowledge. An ACL message according to FIPA [**ACL02**], as illustrated in Table 1.1, describes the state and the actions desired.

1.3.2 Localization, Migration and Execution

Localization Service

Actually, many communication constraints are raised by the mobility feature, since the agents continuously change their execution locations, and must communicate independently of these locations. In this context, it is substantial to provide reliability that ensure communications with mobile objects at any time. Thus, a mobile agent system should offer an agent location service through a names server, which contains the current location of the agent or enough information to locate it. According to Milojičić et al [**Milo99**], several localization schemes were proposed, such as:

— **Research**: the names server looks for the location of an agent according to a defined itinerary, which implies that this later must be known in advance. The number of communications performed may vary from 1 to the maximum number of migrations of the agent (hopes): $1 \leq n_{com} \leq MAX(n_{mig})$

— **Update on the original platform**: the names server located on the original platform of the agents is updated once an agent migrates. In this case, the number of communications performed with the original platform is equal to the number of migrations of the agent: $n_{com} = MAX(n_{mig})$

— **Tracking**: the fact of following a tracking link can locate an agent. Thus, the number of communications performed is at most equal to the number of nodes visited by the agent: $n_{com} \leq MAX(n_{mig})$

— **Recording**: the agents register their actions and movements in a defined and centralized name server, which is not located on the original platform. In this case, the number of communications performed is equal to the number of migrations of the agent plus one communication from the names server to the original platform: $n_{com} = MAX(n_{mig}) + 1$

Migration of an agent

The migration is the process of transferring an active agent from one site to another across a network. According to [**Jain00**], this migration, as viewed in Figure 1.6, considers that the agent may receive messages during its mobility. It includes the following steps:

Table 1.1 – Elements of a FIPA ACL Message

Element	Description
performative	the type of the communicative act of the ACL message
sender	the identity of the sender of the message, that is, the name of the agent of the communicative act.
receiver	the identity of the intended recipients of the message.
content	the content of the message; equivalently denotes the object of the action. The content of any ACL message is intended to be interpreted by the receiver of the message. This is particularly relevant for instance when referring to referential expressions, whose interpretation might be different for the sender and the receiver.
language	the language in which the content parameter is expressed.
protocol	the interaction protocol that the sending agent is employing with this ACL message.
encoding	the specific encoding of the content language expression
ontology	the ontology(s) used to give a meaning to the symbols in the content expression.
conversation-id	Introduces an expression (a conversation identifier) which is used to identify the ongoing sequence of communicative acts that together form a conversation.
reply-to	This parameter indicates that subsequent messages in this conversation thread are to be directed to the agent named in the reply-to parameter, instead of to the agent named in the sender parameter.
reply-with	Introduces an expression that will be used by the responding agent to identify this message.
in-reply-to	Denotes an expression that references an earlier action to which this message is a reply.
reply-by	Denotes a time and/or date expression which indicates the latest time by which the sending agent would like to receive a reply.

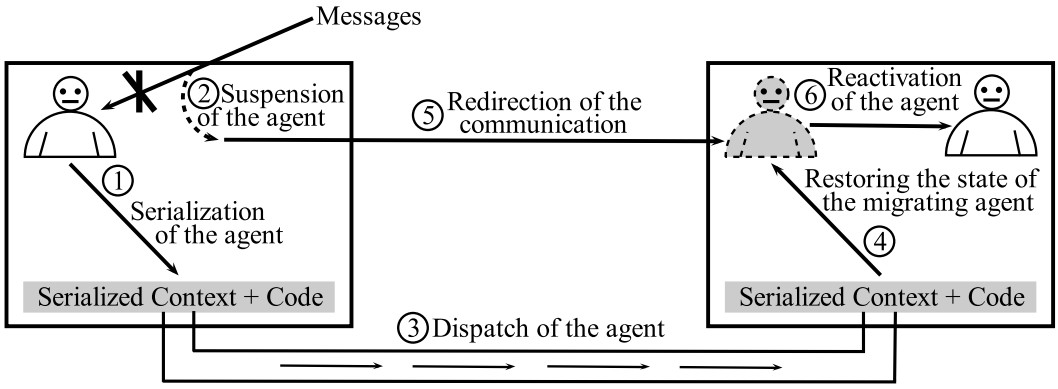

Figure 1.6 – The migration process of an agent between two platforms

1. The serialization of the agent context and the creation of a message to support the serialized context of the agent and its code;
2. The suspension of the agent process, which implies that all communication with other agents are also suspended. Then, the agent is destroyed;
3. The dispatch of the agent's context and code to the destination machine;
4. the state of the migrating agent is restored on the destination machine, where a new and lightweight process is created to pursue the agent execution.
5. The ongoing communications are redirected to the destination machine. Since, the agent is not yet active, these messages will be processed after its reactivation;
6. The reactivation of the agent on the destination machine, once receiving the necessary party of the context for its execution. Therefore, the destination machine checks the dynamic relationship between the agent and its code. At that moment, the migration is terminated and the process on the original machine can be definitely deleted.

The implementing of the agent migration depends on the strategy of processing these steps along with the transferred data. Indeed, there exist two types of migrations:

— **Strong Migration**: it allows an agent to move whatever its execution state. Once arrived, the agent resumes its execution exactly where it was before the migration. The destination of the agent may be decided by the agent itself or by a related party.

— **Weak Migration**: it only transfers the agent code with data. Once on the destination platform, the agent restarts its execution from the beginning, through calling the entry point method of the agent execution of the agent. Thus, the execution context of the agent is reset.

Execution of an agent

The execution of a mobile agent on a hosting machine relies on two factors. The first one is the capacities involved into the mobile agent, and which allow him to be independently executed. The second one concerns the available resources on the hosting machine, and which are used by the agent to accomplish its tasks. During its travel, the mobile agent could visit many machines with various operating systems, and where it may be necessary to get access to the user interface, the file management system, the external applications and the network

interface. Therefore, the big challenge is to maintain the independence of the mobile agent execution regarding the hosting systems, while granting the necessary authorizations to access the system resources.

The mentioned challenge has been overcome through the use of the object-oriented language "Java", for the implementation of mobile code [**Gom01**]. In this latter, the applications are compiled into bytecode and executed in a virtual machine (JVM), which becomes actually a universal execution environment since it is available on all machines in distributed systems. Thus, the programmer has not to manage different languages and versions of a program, according to the systems on which it will run. In addition to the portability feature, Java provides a variety of services that allow the construction of mobile distributed applications, such as:

— The serialization/deserialization of an object in order to transfer it over the network,
— The communication between two objects located on remote machines (RMI, socket, etc ...),
— The dynamic loading of the code from a remote site.

Actually, the diversity of available platforms raises the compatibility issue during the migration between different platforms, where each platform has its own perception to define the behavior of agents. In order to solve this problem, many contributions such as [**Over04**] have proposed the integration of generic languages that describe the agent and allow its reconstruction in its native language.

1.3.3 Security, Fault-Tolerance and Traceability

Security of agents

Security is considered as a critical feature that must be provided in a mobile agent system, since the agent during its trip has to interact with different systems and resources that are characterized with various levels of trust. According to this, two principal categories of security issues are identified:

— **Malicious Platform against Agent**: when an agent is migrating to a new platform, it has to expose in clear its code, status and data. This makes it susceptible to confidentiality and integrity threats on behalf of the hosting platform, which exploits its information and manipulates its behaviors and results. Thus, mobile agents need to be protected at two levels: the protection of the code (changing behavior) and the changing of the agent status. Many approaches are proposed to address this security problems, such as: detection based on tracing techniques, black-box approach and cryptography mechanisms,
— **Malicious Agent against Platform**: a mobile agent could have free and unauthorized access to the runtime environment and thus, it could violate its confidentiality, integrity and availability by intercepting or modifying its data, fully exploiting its resources, cloning or migrating indefinitely. The main two approaches used in this category are: the access control to limit the access to the local resources of the platform, and the authentication where the real identity of the agent is verified.

Fault-Tolerance

During its execution, the mobile agent interacts with several hosts, which exposes it to an eventual failure or unexpected disconnection of the site on which it is running. This failure or disconnection lead to the disappearance of the agent, and subsequently the dysfunction of the application based on it. According to this, an application is said to be secure when it is

able to continue processing in case of failure of a system entity. This property is called "Fault-Tolerance".

Within an architecture of services, various types of failures are considered, such as: *stop* (crash or failure) where no result is returned after repeated invocations, *omission*, *temporal failure* where a correct response did not arrive in defined time interval, *value failure* where incorrect values are received. Concerning the applications based on mobile agents, additional types of failures are depicted in [**Sha06**]:

— An agent may disappear after visiting numerous platforms, and thus the results of its execution will be lost.
— In case that a disappearance of an agent is detected, it is substantial to inform its initiator platform.
— The global and successive execution of the agent on different sites causes a problem of atomicity.

In order to cope with these failures, the majority of mobile agent environments adopt a checkpoint mechanism, which is used to subsequently restore the agent in case of failure. However, during the restoration process, it must be taken into consideration to not have two active agents at the same time.

Traceability

A mobile agent is an autonomous entity that executes in asynchronous manner. Thus, either for the native platform that launches the agent or for the platforms receiving it, it is very useful to have information on the performance of the agent, its state, the machine where it comes and that on which it is located. Hence, despite their autonomy allowing them to choose where to go and when to execute, the use of traceability mechanisms becomes primordial to monitor and control the movement of mobile agents.

For instance, it will be interesting to record that an agent has changed the order of the sites it should visit because of a temporary unavailability of a network link, or that it has waited until this disruption is trailed away. This also concerns the agents whose missions are either exploratory (robots on Internet) or make reference to specific goals and needs (the agents charged to find the right interlocutors to carry out their tasks).

1.3.4 Life cycle and Control

A runtime environment for mobile agents must provide the user with the ability to control the activities of the agent, that goes through several stages in its life cycle. According to El Falou [**Fal06**], these stages are the following:

— **Creation and Initialization**: when an agent is created, it is initialized with the necessary information to its execution, such as the itinerary and its user's preferences. Furthermore, the owner of an agent initializes it with the necessary information necessary to interact with its hosting environment, such as the owner and sender identities.
— **Migration**: the aim of the migration is usually promoted by a local operation with stationary agents or other mobile agents running on the same site or exploiting additional resources (computing power, memory). Thus, before resuming its activities, the agent will need information on the new destination.
— **Activation and Desactivation**: an agent is desactivated by suspending its execution and then saved using serialized mechanism. On the other side, the activation is the reverse operation by which the agent is restored to continue execution.

Table 1.2 – Call-backs in an environment of mobile agents

Agent Life Cycle	**Call-back**
Creation	afterBirth()
Migration	beforeMove() ; afterMove() afterMoveFailed()
Activation	afterResume()
Deactivation	beforeSuspend()
Termination	beforeDeath()

— **Termination**: once its tasks are achieved, the agent is terminated and its execution process is killed. Before its termination the agent needs to deliver a report to its user.

In practice, the developers make use of methods named "Call-back" to define the life cycle of the mobile agent. These call-backs let the agent informed once an event of its life cycle starts, succeeds or fails, which allows to react or refuse one of them. Table 1.2 gives examples of call-backs in a mobile agent environment.

1.4 Standardization Efforts

In network, each platform or site is is designed with respect to a specific architecture and model of agents, or with respect to a specific application domain. This raises a problem of compatibility between different environments for mobile agents, which can not move to a machine that executes a different system than their native system. Thereby, to ensure a high level of interoperability, it becomes primordial to propose a standardization with minimal concepts and functionalities of mobile agent platforms. In this context, we define two standardization efforts: MASIF and FIPA.

1.4.1 MASIF

MASIF (Mobile Agent System Interoperability Facilities Specification) [**Milo98**] is a standard specified by the Object Management Group (OMG), for the system based on mobile agents. It is based on a set of definitions and interfaces, that allow a level of interoperability between different mobile agents. Indeed, MASIF requires an infrastructure primarily based on the following concepts:

— Agents are viewed as autonomous programs working on behalf of an individual or an organization.

— Agents are hosted and executed in a context defined by the agents system, called "place". It is possible that the source place and destination place of a mobile agent, reside in the same system of agents.

— An agent system is a platform that can create, interpret, execute, dispatch and stop the agent. It is associated with an authority that identifies the person or the organization for which it acts.

— A type of agents system describes the agents profile (language, serialization methods, etc) within this system. Thus, an agent moves from one place to another, if its profile is recognized by the destined system of agents.

— The systems of agents with the same authority are grouped in a *region*, which ensures scalability.

The development of MASIF relies on two CORBA interfaces: MAFAgentSystem and MAFFinder, which have been defined at the side of the hosting system, not at the agent side. The MAFAgentSystem interface defines the operations to manage the life cycle of the agent, such as the creation, suspension, resumption and termination of the agent, as well as the transfer and reception of mobile agent classes. While the MAFFinder interface is concerned by the recording and location of places, agents systems and agents. In addition, MASIF defines a security service mainly based on users authentication, mutual authentication of agents systems, authentication and delegation of agents.

1.4.2 FIPA

FIPA (Foundation for Intelligent Physical Agents) **[FIPA02]** is a standardization organization founded in 1996 in Geneva (Switzerland), with the aim to specify software standards, that ensure interoperability between agents and applications based on agents. In general, there are five categories specifications defined by FIPA:

— **Abstract architectures**: they define the abstract entities needed for the development of an environment of agents.
— **Agents communication**: where the ACL messages (Agent Communication Language), the message exchange protocols, and many others concepts are addressed.
— **The transport of agents messages**: this category deals with the transmission and representation of messages, through various network protocols.
— **Agents management**: this category addresses the control and management of agents within and between platforms of agents.
— **The applications**: where examples of application areas, on which can be deployed FIPA agents, are exposed.

In the recent version of FIPA (FIPA2000), there is a focus on the needs and technologies that allow the agent to take advantage of mobility, such as:

— **Life cycle of the mobile agent**: it is illustrate in Figure 1.7, where a new state "transit" is added to the model of stationary agent, with two actions "move" and "execute". To activate the state 'Transit', the mobile agent must initiate the implementation of a protocol of mobility to go to a new system. The action 'Move', initiated by the agent, allows to place it in a transient state. On the other hand, the action 'execute', which is initiated by the executing system, helps to bring out the agent of the transitional state and activates its execution.
— **Protocols of mobility**: it assembles a set of protocols that support various forms of mobility. This particularly concerns the migration, the cloning and the invocation of the agent.
— **Mobility ontology of the agent**: this gives a definition of a set of objects and functions for mobility. Every object contains a set of parameters, such that each parameter is characterized by: a semantic description, a presence, a type (Integer, Word, String, URL, Term, Set or Sequence) and a list of values that can be supported. Similarly, a function is specified by a symbol, a type of agent supporting this function, a semantic description, a domain, a range, and an arity indicating the number of arguments in the function.

Recently, many platforms of mobile agents adopt the standard FIPA, such as JADE (see below) used in this thesis. Concerning the security issues related to mobility of agents, they were not addressed in the FIPA specifications. However, many efforts were investigated in this sens,

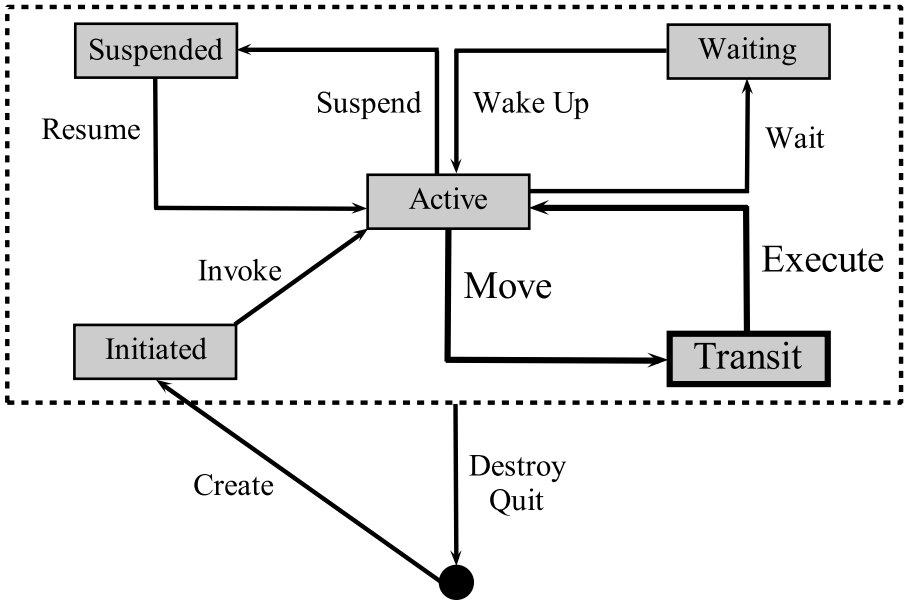

Figure 1.7 – Life cycle of a mobile agent according to FIPA [**FIPA**]

such as in [**Zha01**], where an architecture is proposed with two security services: a service for secure communication that prevents eavesdropping from external network, and another service for a secure execution to prevent unauthorized access to the resources of the agent and the platform.

1.5 Examples of Mobile Agent Platforms

In this section, we expose different existing platforms for the development of mobile agent systems. Not for the purpose of comparing them, but rather to put in evidence the capabilities and characteristics of each one. In general, there is no other rule, in the choice of the platform, than its technical configuration, since a lot of recently added platform were deprecated due to their poor configuration or the complexity to be implanted. We will describe three platforms according to their communication approaches, which are: JADE, Aglets and LIME.

1.5.1 JAVA Agent Development Framework (JADE)

JADE [**Belli01**] is a free project developed in Java by the Telecom Italia Group CSELT1 ,with the cooperation of the University of Parma (Italy), and it is distributed by Telecom Italia Lab (Tilab) under LGPL. It follows the specifications of the FIPA Standard, and provides interoperability with a complete set of agents and services: naming service, yellow pages service, parsing service and a library of protocols interaction, that make the development and set up of the multi-agent systems more easier. JADE platform can be spread on multiple servers, with only one java virtual machine (JVM) executed on each server. Conceived as an agents container, the JVM allows multiple agents to be launched in parallel on the same server.

Being a FIPA compliant agent platform, JADE includes three mandatory components: Agent communication Channel (ACC), Agent Management System (AMS) and Director Facilitor (DF), as illustrated in Figure 1.8.

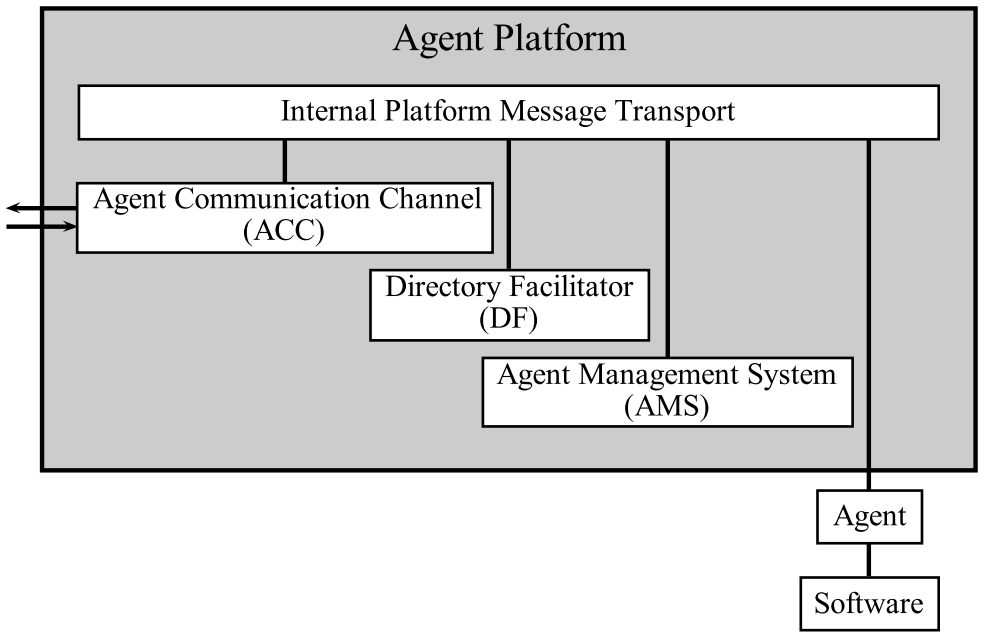

Figure 1.8 – Structure of JADE platform compliant to FIPA

— *Agent Management System (AMS)* provides the naming service, ensuring that each agent in the platform has a unique name. It performs several "management tasks" as well, such as creating and killing agents on containers. It is a mandatory agent providing a directory service as it maintains a register of description of all agents in the platform.
— *Director Facilitator (DF)* provides a yellow pages service. Agents of the platform register with the DF. This allows one agent to trace other agents that provide the services it requires by requesting the DF.
— *Agent Communication Channel (ACC)* is the agent that supports the communication between agents inside and outside the platform. It listens for remote invocations. When it receives an ACL message encoded as a string, which is usually the case of non-JADE agents, it parses the message and converts it into a Java *jade.lang.acl.ACLMessage* object used by all agents in the JADE platform.

The communication among agents within this framework is performed via FIPA ACL messages (Agent Communication Language) sent over an MTP (Message Transport Protocol). The later, works such a chameleon mechanism, that adapts to every situation by transparently choosing the best available protocol. The main protocols used are: RMI, HTTP and IIOP, but there are others that can be instantiated.

JADE adopts a multi-threading method to manage the cooperative and cyclic behaviors, and makes use of a graphical user interface (GUI) for remote management agents (RMA). The GUI creates and runs an agent on a remote server, that needs to be provided with an agents container, already in running mode. This is achieved using some sophisticated graphical tools

that support the correction phase, such as:

— *Dummy Agent*: it displays the messages exchanged among the agents, validates their integration in the SMA and facilitates the debugging.

— *Sniffer Agent*: when the user wants to monitor an agent or group of agents, the sniffer is then used to track the exchanged messages. The screened messages are displayed or stored for subsequent analysis.

— *Introspector Agent*: it allows the control and the supervision of the current agent's life cycle and its behaviors, traced in its exchanged messages.

We make use of this platform for the implementation of our approaches, as it fits our needs for mobility, through its communication mode based on FIPA ACL, that supports serialized objects, which lighten the migration process. It also provides a yellow page directory, where referenced services are integrated and managed using ontologies. Based on behavioral attitudes, the conception of mobile agents allows the achievement of distributed applications.

1.5.2 Agent Applets (Aglet)

Aglets [**Tai99**] are components developed by IBM at Tokyo in 1995, in order to furnish a uniform platform for mobile agents development in heterogeneous areas, such as network and internet. In reality, they are mobile Java objects acting as mobile agents, so, they can move from one host to another, where they are able to begin or continue the execution performed on the last host.

The communication of Aglets is based on the exchange of messages according to KQLM language (Knowledge Query and Manipulation Language). The main components involved in the communication of Aglets are:

— **Aglet**: it is a mobile Java object able to move across hosts able to receive and execute agents. It is an autonomous and reactive object, since it can continue execution once arrived to destination, and dynamically react to the incidents of its environment.

— **Context**: it is the Aglet's executing environment, which is provided with mechanisms and capabilities to control, update and track the aglets, in a uniform manner.

— **Host**: it is every server or machine able to receive one or multiple contexts. It usually refers to a network node.

— **Proxy**: every Aglet owns a specific proxy as delegate, that acts as a shield to protect the public methods of the Aglet from unauthorized direct access. For that reason, it hides the real location of the Aglet.

When an Aglet needs to send a message, it must go through the proxy of the recipient. Each Aglet holds a messaging manager, that allows it to deal with messages one by one in their order of arrival. Aglet is endowed with a great simplicity of implementation, and its security features are based on Java APIs and a personalized controller, that allows users to edit their own preserving-privacy methods. However, the fact of hiding the localization of an Aglet may raise many flaws, especially in environments that are not fault-tolerant, and where tracing the movements of the agents is essential. Concerning dynamic environments, it becomes more narrow and complicated, to define granted rights and rules to access local resources or execute tasks.

1.5.3 Linda in a Mobile Environment (LIME)

LIME [**Murph06**] is a model and middleware, supporting the development of applications that exhibit physical mobility of hosts, logical mobility of agents, or both. LIME adopts a

coordination perspective inspired by the work on the Linda, which is a model of parallel programming, regardless of the place, data or time. Linda is composed of two explicit notions: "tuples" and " primitive access methods" (which: are out(), eval(), in() and read()).

Indeed, LIME is not viewed as a complete platform, since it does not support the various elements described in the standards, but only exploits coordination in order to build distributed applications. Making use of shared storage space called "tuples", the communication is decoupled in time and space, which notably fits the mobile environments where the interlocutors move in dynamic manner. Accordingly, each mobile agent is associated to a "tuples" space, and each host detains an Interface of "Tuple" Space (ITS), at which the mobile agents may subscribe, through providing the tuples to be shared. ITS is the union of tuples shared by local agents. When an agent wants to communicate with one or multiple agents, it disseminates a "tuple" in space.

Table 1.3 – A comparative table of the defined agent-platforms: JADE, Aglets and LIME

	Agents		**Metrics**		**Communication**		
	Type	**Mobility**	**State**	**DF**	**Method**	**Type**	**Protocol**
JADE	Proactive Reactive	Weak	Yes	Yes	Asynchronous	Asynchronous Synchronous	RMI HTTP IIOP
Aglets	Proactive	Weak	Yes	No	Asynchronous Synchronous	Asynchronous Synchronous	ATP RMI CORBA
LIME	It supports different types of agents, depending on the hosting system, through a synchronization medium		Yes	No	Synchronous	Asynchronous Synchronous	Sockets

This mechanism is extended for use in remote communications. For that purpose, a set of several ITSs is assembled in a Federation of Tuples Space (FTS), which is managed by a leader site that adopts a conventional election mechanism, whose role is to consider the incoming and outgoing ITSs. This allows the agents to interact remotely and be warned of any changes, in order to adapt and maintain the optimal configuration of their environment.

Although its simplicity and its flexibility, the centralized management provided by the FTS within this model, cannot be adequate for highly dynamic environments. This is more illustrated in cases where the leader of a FTS is passed away, or when the communication links between agents, asking for remote synchronization, are not constant.

Finally, we have grouped the main features of these Java platforms in the summary Table 1.3, in order to get a quick overview.

1.6 Conclusion

This chapter was devoted to define the principal concepts and aspects in the mobile agent systems. After discovering the evolution of communications and services over the network until mobile agent paradigm, we have provided the usual definitions and fundamentals in this field, its advantages as well as an overview of its application disciplines. Subsequently, the essential services required for the execution of mobile agents are described, with a highlight on the admitted standards and examples of commonly used agent platforms.

Chapter 2

Security of Systems based on Mobile Agents

Contents

Certainly, mobile agent paradigm provides promising solutions that facilitate the distributed execution through open networks. However, it introduces serious security problems mainly related to the luck of confidence. Thus, in order to implement applications based on mobile agents, in a such way they reach their full potential, it is strongly recommended to employ them in mobile agent platforms which are safe and secure.

In this chapter, the security issues associated to the use of mobile agent technology are emphasized. We begin by exposing the substantial requirements needed to maintain the security of an information system in Section 2. Then, Section 3 enumerates the famous attacks leaded separately against the mobile agent and the agent's platform. In Section 4, the countermeasures adopted to protect the mobile agent and the agent's platform are investigated. Finally, a synthesis is provided in Section 5.

2.1 Security Requirements

In general, the development of an application necessitates the definition of requirements, whether they are functional describing the tasks to accomplish, or non-functional defining the way these tasks are performed. The security is among the non-functional requirements, which represents a great challenge to the expansion of new technologies. It is often related to the

design and requirements identification of the involved system, without compromising the proper functioning of the application.

Attacks that occur in a systems based on mobile agents may have different sources and different targets. Indeed, there are four classes of attacks: two classes whose source is a malicious mobile agent, and the other two are caused by a dishonest agents system. In order to resist to the different forms of attacks in an information system, it is of a paramount importance necessary to introduce mechanisms that ensure the security needs [**Bra05**], such as authentication, confidentiality, integrity, availability, non-repudiation and access control.

— **Authentication**: authentication of an information system is the process to verify the identity of another entity (person, computer ...) to allow its access to resources (systems, networks, applications...). It permits then to validate the authenticity of the entity in question. The fact that a mobile agent authenticate a platform is the first and essential resource before it proceeding to its execution, because a malicious site could hide its identity in order to attract the agent, and then leads attacks on it such as masquerade balls and cloning agents. Similarly, a mobile agent has to authenticate on each visited agents system, and therefore, an agent system is then capable to decide if it is a trusted agent.

— **Confidentiality**: data confidentiality consists on ensuring that information is kept secret and that only authorized persons may have access to it. It is a very recommended property in elaboration of a security policy or approach. Maintaining the confidentiality of the stored data, the parameters of treatment, and any information circulating through the network is the main objective of the security. This concerns mainly the private data such as the resources and services of the agents system, as well as the code and data of the mobile agent.

— **Integrity**: the integrity is the principle made to ensure that the data are those believed to be, and verifying data integrity consists in checking whether the data have been altered during transmission (incidentally or intentional) or not. It is a very valuable property to maintain confidence between the communicating parties. For example, in the context of mobile agents, the itinerary of the agent is a sensible data that requires specific protection against all forms of tampering.

— **Availability**: availability ensures the proper and not abusive use of the services and/or resources of the system. In addition, this property provides access to resources and/or services as long as the requesting entity is authorized. Both the agent and the agents platform have to ensure the availability of their data and services to local and remote agents, and provide controlled concurrency, support for simultaneous access, deadlock management, and exclusive access as required. They are also concerned by detecting software/hardware failures, and supporting fault-tolerance and fault-recovery.

— **Non-Repudiation**: it is the assurance that the originating entity can be held responsible for its communication. In other words, availability requires that either side of a communication cannot deny the communication later. Thus, when a mobile agent is executing on an agents platform, this execution, including all data involved, operations performed and results obtained, cannot be repudiated by one of them. This is due to the fact that, the important communication exchanges are logged and recorded, basing on the identities authenticated, so as to prevent denials by any party of a transaction.

— **Access Control**: this means the concept of granting or denying access rights to a user, program or process. Furthermore, access control requires that only legitimate users have rights to use certain services or to access certain resources, for which unauthorized users are kept out. Thus, both agent and agents platform need to adopt a security policy based

on access control, to prevent their resources from the abuse of malicious entities, that attempt to get unauthorized access. Here, there are many mechanisms to decide whether or not to grant a request to an entity, such as: password access, digital signature, etc.

— **Accountability**: Both mobile agents and agent platforms need to be held accountable for their actions. Mobile agents need to be held accountable for services and data it accessed, and platforms need to be held accountable for the services and data it provided. These actions need to be uniquely identified, authenticated, and logged. This log, maintained by either or both the mobile agent and agent platform, must be tamper-proof and non-repudiable. Measures need to be in place to handle situations when the log becomes full.

2.2 Security Threats

In general, it is difficult to decide in advance the threats and vulnerabilities you may face in a system, since the probability of risk occurrences changes depending on variety of system metrics, including its nature and status. Thus, it becomes primordial to recognize these risks and try to prevent them, through implanting an effective and reliable security policy.

An attack is viewed as the set of environmental actions and behaviors, that may threaten a system and cause serious damages. There are two types of attacks:

— *Passive Attacks*: in this attacks, the data and resources of the system are not modified, only their confidentiality is affected. Examples of passive attacks are: eavesdropping, illegal copying, etc.

— *Active Attacks*: they are more potent, because they are able to alter the information or remove it, as well as influence the overall system behavior. Examples of active attacks are: alteration, denial of service etc.

In literature, there are many classifications of attacks against mobile agent systems. In addition to the technical report of the NIST [**Jan99**], we were inspired from [**Alf05**; **Bier02**] to distinguish two major categories of attacks as illustrated in Figure 2.1 attacks led by malicious agents on platforms or other agents, and attacks led by malicious platforms on the hosted agents.

2.2.1 Attacks of Malicious Agents

When a malicious agent is moving across several platforms over the network, it may exploit abusively the resources and services of these platforms, in order to produce severe harms, or communicate improperly and unsafely with other naive agents encountered in the itinerary. The attacks of mobile agents can take different forms, as described bellow.

Masquerading Attack

A mobile agent is masquerading when it hides its real identity and uses the identity of another agent, in order to get privileges, for which it normally does not have the right. This may cause serious damages to the hosting platform and accuse another agent to be responsible of this damages. Consequently, many trust relationships within a community of mobile agent systems are broken.

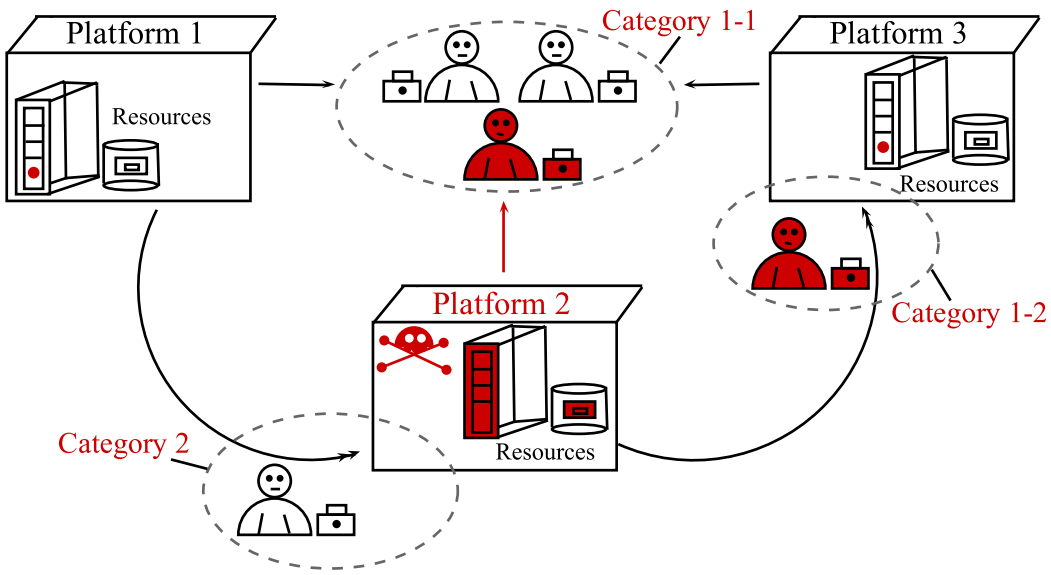

Figure 2.1 – Categories of Threats in Mobile Agent Systems

Denial of service Attack

A denial of service attack may occur either intentionally or through a programming error, when an agent executing on a hosting platform, attempts to excessively consume the resources of the platform (CPU, bandwidth, ...). The big concern of this attack relies on the fact that the agent code is elaborated outside the platform charged with its execution, and this code may be malicious.

Unauthorized Access

Access control is a mechanism, that aims to manage and monitor the access of legitimate users and processes, to the services and resources for which they are authorized, as it is specified in an adopted security policy. Thus, each mobile agent has to respect the security policy of the visited platform. However, when this agent comes to obtain illegally privileges and services, it may harm the hosting platform and other local or remote agents.

Repudiation

Repudiation is a breakdown of the communication among the entities of the system, caused by an agent involved in the transactions. It may lead to huge problems and conflicts, either it is produced accidentally or intentionally. Thus, protecting the platform from agent repudiating transactions is considered to be substantial and crucial, through introducing the appropriate security mechanisms and countermeasures, taking into account the techniques to solve the occasioned disagreements.

2.2.2 Attacks of Malicious Platforms

Since an agent platform is able to execute a mobile agent, thus, it may have malicious attitudes and attempt in various ways to harm the mobile agent. Among the attacks in this context, the following ones are identified.

Eavesdropping Attack

Eavesdropping is a conventional attack also called "Listening", as it intercepts secret communications. mobile agent platforms are considered as prolific environments, where this kind of attacks may occur and expand. This is due to the fact that these platforms take control of the received agent exchanges, including every executed statements or produced information. Moreover, even when the mobile agent adopts robust security mechanisms to be protected, the eavesdropping platforms can lead analysis on the agent's behaviors, in order to approach its approved strategies and recognize the services it may provide.

Alteration Attack

Alteration attack occurs when an agent suffers from lack of its data integrity. An agent that moves to a client platform must expose its code, state, and data, in order to be executed. Thereby, the platform may modify the own data of the mobile agent, as well as other data gathered and obtained on previously visited platforms. What is worst is when the malicious platform alters the execution state of the agent, either temporarily to force a modified execution according to the platform specifications, or permanently through injecting viruses, worms or Trojan horses to attack other platforms being in the itinerary of the mobile agent.

Masquerading Attack

A malicious platform can hide its real identity and masquerade as another trusted host, in order to deceive the mobile agent as to its true destination and then extract its sensitive information. What is more harmful in this attack is that it can facilitate other attacks, such as: eavesdropping, abusive extraction of sensible data, etc. In addition to deceiving naive agents, the masquerading platforms can harm the platforms whose identities were exploited.

Denial of Service Attack

Once arriving on an agents platform, the mobile agent waits that its requested services will be properly executed on the defined and allocated resources, according to specific constraints. However, the malicious platform may ignore the services requests of the agent, introduce unacceptable delays for critical tasks, even more, do not execute the mobile agent or terminate its session without any notification, which could lead the agent to be dead-locked.

Incorrect Code Execution

In this kind of attacks, a malicious platform may change the way it executes the code of the agent, without modifying either its code or its execution state. In this context, the platform can assign or return false values during computations and comparisons, re-execute the agent to copy some of its parts or messages, or simply re-execute the agent with different data.

2.3 Protection of Mobile Agent Systems

The security problem in the mobile agent systems represents a brake on the actual use of this technology. The latter requires, in one hand the protection of the resources and data on the host machines, through limiting the access rights and consumption of resources, and on the other hand, the preservation of agent's integrity and confidentiality as well as those of its communications [**Alf05**].

Indeed, mobile agents become a new field of investigation for the security research field. According to this, there are two main areas of research: the protection of agents against malicious platforms and the protection of agent's platform against malicious agents [**Alf05**; **Bella04**].

2.3.1 Protection of the Mobile Agent

Cryptographic Traces

Cryptographic Traces or "Execution Tracing", as proposed by Vigna [**Vigna98**], is a detection technique based on a post-mortem analysis of data and events, which generate "traces" to notice any abnormal execution on the hosting platform, either by the agent or the platform itself. It enables an agent's owner to check the agent's execution history (logs of the actions performed by the agent), and see if it contains any unauthorized or improper modifications done by a malicious platform.

A trace is represented by a pair (n, S), where n is the set of the unique identifiers of the operations performed, while S is the set of signatures containing the line codes of these operations. Indeed, the statements in the agent's code are divided into two categories: "white statements" where only the values of the agent's internal variables are used, and "black statements" which require information from external environments. Thus, only the black statements are considered for signature in S. Table 2.1 shows an example of a trace.

Table 2.1 – An example of a trace

Code	**Trace**
read(x)	(1, x=30)
a = b+ c	(2, none)
e = encrypt(input)	(3, e = "qyt2b0#")
T = T + e	(4, none)
isAuthenticated(param)	(5, "True")

This technique assumes that all entities (agents or platforms) in the itinerary of the agent communicate through using signed messages, which implies that they must hold a private and public key to compute the digital signatures. When a platform receives an agent, it produces an associated trace during the execution of the agent. This trace may include information such as: the unique identifier of the message, the identity of the sender, the timestamp, the final state as well as a trusted third party to be consulted in case of conflicts. Once the execution is terminated, the trace is signed by the hosting platform using its private key, and then transferred along with the agent's code and state to the next destination. In the return of the agent to its owner platform, this latter may suspect that a certain platform cheated while executing the agent. In this case, the owner can either re-execute the agent from its initial state or ask

the suspicious platform to reproduce the trace. Subsequently, the agent's owner compares the reproduced trace with that originally provided by the suspicious platform. The same principle is used to protect a platform against malicious agents.

This first version of execution traces has shown some limitations, such as the potential large size and number of logs to be retained and the fact that the owner platform needs to wait until it obtains suspicious results in order to run the verification process. According to this, Tan and Moreau [**Tan02**] have proposed a new version where the verification process of the trace is assigned to an independent "Verification Server" as a trusted third party. This verification server receives a copy of the agent before its migration. Afterwards, each visited platform forwards the produced trace to this server, that simulates the execution of the agent on that platform using the agent's copy. This process is repeated until the return of the agent to its home machine.

Partial Result Encapsulation

This technique consists of detecting the modifications applied to the agent, through encapsulating the results of the agent's execution performed on each visited platform. Thus, the encapsulated information is lately verified to discover any security breaches. This verification, to ensure agent's results validity, can be evaluated on the native platform or on an intermediary placed in the path of the agent. Indeed, the encapsulation aims at preserving many security requirements, such as the confidentiality, integrity, non-repudiation and accountability, through using cryptographic primitives, namely encryption, digital signature, hash function and authentication code. In general, there are three manner to encapsulate results: with the agent, with the platform or with a trusted third party.

Among the approaches proposed in this context, Young et Yung [**Young97**] defines a method called "sliding Encryption", based on the public-key cryptography to encrypt small amounts of results whithin a larger block and thus obtain small piecess of ciphertext. This aims to reduce the volume of encrypted data, which fits the limited storage capacities of the agent. In addition, Roth [**Roth98**] presents an oriented-platform method, where each platform executing the mobile agent adds the obtained results to the chain of results included in the agent, through linking them to the results on the previous platform. In other words, a platform has to digitally sign its results with its private key, and then hashes the concatenation of the chain of results with its encrypted results. The fact of encrypting the results with the private key of the agent's owner, ensures strong integrity.

Nevertheless, the technical report of the NIST [**Jan99**] describes many flaws and limitations of this technique. For instance, if a platform maintains the set of the keys, it will be possible for this platform, once visited again, to partially alter the results without being detected.

Code Obfuscation

Code obfuscation is a technique where a transformation is enforced to the agent's code, before being sent for execution on different platforms, with various trust levels. According to a predefined security policy, this transformation must preserve the behavior of the mobile agent and the actions it is charged to perform. That is to say, a platform that hosts an agent with an obfuscated code, could neither disclose the sensible information included in the code, as they become hard to understand and analyze, nor modify the attitude and behavior of the agent's execution.

In literature, many obfuscation transformations were proposed [**Wro02**], including: "Data Obfuscation" where only data and data structures are concerned without modifying the code

itself, "Layout Obfuscation" where information not affecting the code execution are modified or removed, such as comments and debugging lines, "Control Obfuscation" where the control flow used by the code is altered, and "Preventive Obfuscation" that focuses on preserving the code against the decompilators and debuggers.

Hohl [**Ho98**] makes use of this technique to constitute a time-limited-black box, where the mobile agent can be involved and safely executed on suspicious platforms, for a limited period. However, according to an analysis and critical study provided by [**Ann03**], the obfuscation technique combined with reverse engineering, can be suitable for delaying the threats on agents, but not preventing them. Eventually, despite the fact that this technique resists to the impersonation and denial of service attacks, the real concern is to implement it in practice.

Computing with Encrypted Functions

This technique is a software solution based on cryptographic primitives to preserve integrity and privacy of the mobile agent's code. The main concept of this technique consists in incorporating an encrypted function inside the agent's code, that will be executed on a mobile agent platform, which totally ignores any substantial information about the function.

Sander et Tschudin [**San98**] proposed a technique that conceives the mobile agent as a black box for the platform where it is executing. Thus, they force the platform to execute the agent's code, without being able to decrypt the encrypted function involved in it. This is formulated as follows:

- **Problem**: Alice has an algorithm to compute a function f. Bob has an input x and can provide a service to Alice by computing $f(x)$. However, Alice does not want Bob to learn anything about the function f. Besides, Alice and Bob could not communicate or exchange messages during the computing of $f(x)$.
- **Solution**:
  - Alice encrypts f to get $E(f)$;
  - Alice creates a program $P(E(f))$ that implements $E(f)$;
  - Alice embeds the program $P(E(f))$ within the mobile agent and sends it to Bob;
  - Bob executes the mobile agent, which implies executing $P(E(f))$ to x to produce $P(E(f(x)))$;
  - Bob sends back the mobile agent to Alice;
  - Alice decrypts $P(E(f(x)))$ to get $f(x)$.

Although this technique enables the execution of encrypted programs on untrustworthy platforms, but it is still limited in employment, as it is applicable only for polynomial and rational functions for which a suitable encryption could be known. Finally, it is worth to mention that the computing with encrypted functions is vulnerable to some class of attacks such as denial of service and replay attacks.

Environmental Key Generation

The technique of environmental key generation as proposed by Riordan and Schneier [**Rio98**], makes an agent generates a key when an environmental condition is true. This key can be used to decrypt a message destined to an agent, whose sender wants it to be readable only under certain conditions, such as matching a certain search string. It is primordial that the platform ignores the significance and dependence of these conditions, because they master the whole environment. That is to say, if the environment does not satisfy the conditions to generate the key, it is impossible to guess the function of the agent.

The generation of the environmental key could be performed in different manners:

— using fixed data on a channel, such as web pages and severs of news. However, the utility of this method is based on the entities that have direct or indirect access to the channel, and which could manipulate the data.
— using temporal constraints, such as generating a key only before or after a specific date, or according to a fixed time interval.

This technique may be of great benefit for many applications other than mobile agents, such as remote alarms, blind search engines and directed viruses. However, it stills suffering from some limitations, mainly related to the maliciousness of platforms that come to force the mobile agent to execute different functions, even the required conditions to generate the activation key are met.

Spatial Security Policies

A Sentient Computing environment as defined by Scott et al [**Scott04**], is an environment where the involved system may perceive his own state and use the related information to customize its behaviors. The authors of that research work initiate a technique to model new security policies for mobile agents, mainly based on agents location. These policies called "Spacial policies" are employed to make the location assertions (either positive or negative) more dynamic.

Regarding future perspectives on sensitive applications to locations, these dynamic assertions will be of great interest and utility. This is due to the fact that they can refer to the location of physical and virtual objects in the world. This technique provides a useful way to compel the mobility of agents, in order to safely use the mobile agent technology and to simplify the development process of future sensitive applications.

Self-Executing Security Examination

Abbreviated as "SENSE", self-executing security examination was proposed by Page et al [**Page04**] as a general software architecture for mobile agent protection. This architecture includes security schemes, involving programmed mobile agents that carry their own security implementations, make them more independent.

The basic idea behind this concept focuses on the use of a scan algorithm by the mobile agent, in order to verify the integrity of its code in arbitrary temporal intervals. Thereafter, the returned scan result is compared with a code image carried by the agent. Thus, the mobile agent can easily detect any attempted attacks and ensures its integrity without having to consult the owner host.

2.3.2 Protection of the Agents Platform

Agent Authentication

The conception of mobile agent environments is similar to distributed systems, where authentication is a big concern. In case of mobile agents, in addition to verify the integrity of the code and data of the agent, it is strongly needed to authenticate platforms and servers working on behalf users.

Among the various tentatives in this context, Greenberg et al [**Green98**] proposed to digitally sign the mobile agent with a public key algorithm, which allows to generate a certificate that ensures agent's integrity, as well as to authenticate the agent's owner and receiver. Moreover, confidentiality is also preserved through encrypting the mobile agent, so that only the

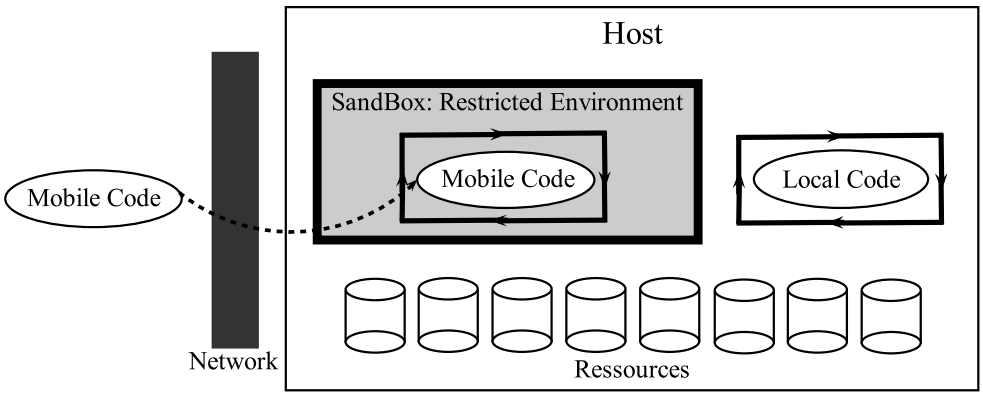

Figure 2.2 – Sandboxing Technique

destined platform decrypt it. However, this method expresses many limitations [**Jan99**], particularly in the distribution of the public keys. It is substantial to find a mechanism to easily and promptly the public key certificates of users, namely when the number of users is increasing. Another limitation is the classification of the certification related to a program, while this later could not be usually defined as inoffensive. In addition, the verification made in a short and limited time could not be effective and qualifies the state of the mobile agent while executed on malicious platforms.

Sandboxing

Sandboxing [**Vigna03**] is referring to the sand boxes used by minesweepers to detonate devices safely, as illustrated in Figure 2.2. This technique allows the execution of the mobile agent's code in a restricted environment (sandbox), that appears similar to the global system, and where restriction affects certain code operations such as: the access to the file system, the opening of a network connection, the access to the properties and programs of the local system, or invoking programs on the local system. This ensures that a malicious mobile agent cannot cause any harm to the execution environment that is running it. It mostly depends on the access control of the resources and the security policy associated, and proceeds to the execution of a mobile agent with limited privileges, even suspicious, in the sandbox without caring about security issues.

Among the tools that facilitate the implementation of sandboxing, there is the code interpreters [**Haus00**]. Those later involve three principal components: "*ClassLoader*" that converts the remote code in data structures able to be added to the hierarchy of the local class, "*Verifier*" that performs a set of checks, before charging the remote code, to ensure that only legitimate code are executed, and finally, "*Security Manager*" controls the operations performed by remote classes. However, this technique suffers from various troubles. When one of the mentioned three components fails, the security could be violated, because an incorrect classification of a remote class as a local one, allows it to exploit the privileges of the local class. Another issue is the increase of the execution time of a legitimate remote code, but this can be overcome by Code Signing technique.

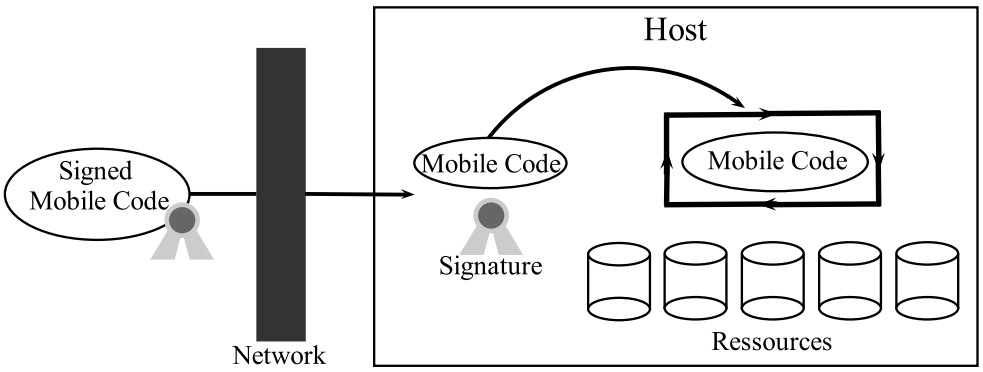

Figure 2.3 – Code Signing Technique

Code Signing

When a mobile agent is created, the *Code Signing*, as shown in Figure 2.3 incites his owner to digitally sign it so that it can be identified during his round-trips. It enables every visited platform to verify that the agent's code has not been modified since it was signed by its owner. Thus, this technique allows to obtain a high-level authentication for hosting platforms, as well as ensure the integrity of the code.

Digital signature and one-way hash function are two principal primitives used by code signing. This later was remarkably introduced by Microsoft to elaborate the Microsoft Authenticode, which is typically used in the framework ActiveX. Java JDK 1.1 for controls and Java applets [**Madison03**; **MSDN**]. According to this, an applet with a valid signature is executed as a reliable code and therefore allowed to get access to all available properties in Java, while an unsigned applet or whose signature is invalid is executed in a sandbox. Moreover, digital signatures widely benefit from the public key infrastructure, because the certificates containing the identity and the public key of an entity can be easily located and checked. The main problem of this technique is that it verifies the identity of the code creator, but it does not guarantee that it is trustworthy. This cloud absolutely not solve the fundamental risk associated with the agent behaviors, which leaves the hosting platforms vulnerable to damages due to faulty or malicious code from a reliable source. In addition, a malicious mobile agent not only exploits the granted privileges to harm the executing hosts, but also to open a door for other malicious agents by changing the acceptance policy on the platform. Another issue makes this technique overly restrictive towards mobile agents belonging to untrustworthy entities, as they do not run at all. Consequently, rather than relying on the identity of the code creator, it would be more efficient to have a code verification conducted by a reliable party or by an evaluation service.

Access Control

Every program running on an information system needs to get access to the resources in order to perform its task. Therefore, for security concerns, the environment executing the mobile agent must be restricted, through strictly controlling the resources needed for accomplishing jobs. This mechanism is approximately an extension to the Sandboxing, such that a more complex access control policy is implemented, and where agents could not interfere with other agents or platforms to establish separated isolated domains and control all cross-domain accesses. According to this, a concept called "Reference Monitor" [**Shnei01**] was proposed as a

mediator for all accesses, which gives permissions to a mobile agent for a number of resources according to the result of the authentication. It is based on a wide variety of conventional security mechanisms, such as:

— Mechanisms that isolate processes and controls;
— Cryptographic primitives to encrypt the exchanged information, as well as to identify and authenticate users, agents and platforms;
— Mechanisms to control access to the resources of computations;
— Mechanisms for audit and security of events occurring at the agent platform.

Examples of reference monitors are the principal interpreter of SAFE-TCL and the "Security Manager" in Java. This technique solves the problems of Sandboxing as it provides a finer granularity on the protection and authorization, as well as an adaptation of access rights according to the mobile agents.

Proof Carrying Code

Code verification is a principle that analyses the semantics of the agent's code, basing its structure and behavior, and according to a given security policy. In addition to Sandboxing that ensures rudimentary and expensive verification, many other approaches are proposed. Proof Carrying Code (PCC) is one among these approaches that promote the automatic verification of the code before its running, based on a conformity proof. PCC was firstly introduced by Necula and Lee [**Nec98**] to allow a system receiving a code from another system to determine with certainty and instantaneously if this code is safe to install and run. The idea is that the code producer gives an encoded evidence or proof that the code adheres to the security policy of the consumer. The proof is conceived such that it can be digitally transmitted to the consumer, and that it can be quickly validated through a reliable, simple and automatic procedure. Once the code has been verified, it may be executed safely. PCC guarantees the safety of the incoming code providing that there is no flaw in the verification-condition generator, the logical axioms, the typing rules, and the proof-checker. Many advantages of PCC are highlighted. First of all, it is "tamper-proof" as any attempt to modify the code or the proof will be detected. It is also "self-certifying", because no cryptography or trusted third party is required. Besides, it involves a static program with low-cost checking, after which the program can be executed without any expensive run-time checking. However, PCC shows some limitations, that include the potential size of the proof and the time consumed by the process of validating the proof, as well as the proof generation which is only possible for simple properties such as safety memory, and very difficult concerning complex properties.

State Appraisal

During the mobility of the agent across multiple platforms, it carries its code, data (static or gathered) and execution state. This last gets changed and produces dynamic data while the agent is executed at each host. State Appraisal was introduced by Farmer et al. [**Farm96**] to guarantee that an agent has not become malicious or modified due to the alteration of its state by a malicious host. This technique is based on the use of appraisal functions, which determine the privileges granted to an agent based on conditional factors and invariants. These functions are used by platforms to verify the execution state of an incoming agent, and determine the privileges that this agent can get in order to proceed to its execution. According to this, the author of a mobile agent must anticipate the possible modifications to the agent's state and to counteract them within the appraisal function. Likewise, the agent's sender generates another appraisal function to specify the set of permissions to be requested by the agent. Both author

and sender of the mobile agent applies constraints on the state to reduce responsibility and/or control costs, and they sign the agent to protect their appraisal functions against non-detectable modification.

State Appraisal provides a flexible way for an agent to request permissions, according to its current state and the task needed to be executed on a particular host. However, its major issue consists in the difficulty to formulate suitable security properties for the mobile agent, and to obtain a state appraisal function that guarantees these properties. Thus, many attacks still difficult to detect because they were not predicted, so they are not included in the appraisal functions to ensure the necessary protection.

Path Histories

The basic concept of path histories approach is to maintain an authenticable recording of the platforms already visited by an agent [**Reis00**], since this agent during its travel life may be hosted by multiple platforms with various trust levels. Actually, when the mobile agent migrates to colossal number of platforms, it faces a great difficulty to maintain trust. Moreover, an agent whose itinerary is not decided in advance, could not be easily confided, such an agent that is looking for specific information and which updates its travel path dynamically. In order to build a path history, each platform visited by the mobile agent must append a signed input, that contains the current platform's identity and that of the next platform to be visited by the agent. According to the information involved in the history, the current hosting platform decides whether to execute the agent or not and what privileges and services should be granted to him. In addition, so as to prevent tampering attacks, each signed record constructed by a platform should include the record of the previously visited platform. This technique does not prevent the platform from malicious behaviors, but applies a deterrent effect due to the non-repudiation of the platform's record signed basing the path. However, the main problems with this technique consist in the fact that the path verification process becomes increasingly expensive with the increase of its history. Moreover, it depends on the ability of the platforms to correctly judge the reliability of those previously visited.

2.4 Synthesis

In this chapter, we have approached the security issues in the systems based on mobile agents. According to the security requirements commonly requested in each information system to maintain its safety, we have exposed the related threats to each category and the counter-measures to be considered in order to prevent them. Table 2.2 provides a synthesis of this chapter.

Table 2.2 – Synthesis of the security attacks and their counter-measures

		Counter-Measures	**Category**
Integrity Attacks	Alteration	Encrypted Funcions Environmental Key	Prevention
		Cryptographic Traces Partial Result Encapsulation Code Signing "SENSE"	Detection
	Unauthorized Access	Access Control	Prevention
		Cryptographic Traces	Detection
Availability Attacks	Denial of Service	Access Control	Prevention
		Cryptographic Traces	Detection
Confidentiality Attacks	Eavesdropping	Encrypted Functions Environmental Key Code Obfuscation	Prevention
	Unauthorized Access	Access Control Sandboxing	Prevention
		Cryptographic Traces Partial Result Encapsulation	Detection
Authentication Attacks	Masquerading	Agent Authentication Sandboxing	Prevention
		Path Histories Code Signing	Detection
Accountability Attacks	Incorrect Code Execution	Sandboxing	Prevention
		State Appraisal Proof Carrying Code Partial Result Encapslation	Detection

Chapter 3

Serialization and Cryptography for Mobile Agents

Contents

In this chapter, we attempt to address the security issues related to the mobility of an agent. For that purpose, we introduce two new approaches to secure the mobile agent during its trip, in which it may be hosted by malicious platforms or interact with other agents qualified as suspicious. In these approaches, we make use of serialization mechanism in its two forms: XML-based and binary. Besides, to establish an authentication mechanism between the agent and its host or interlocutor, as well as to ensure its confidentiality and integrity, cryptographic primitives are employed such as: the key exchange and key agreement protocols, signatures, encryption algorithms, etc.

3.1 Preliminaries

The communication between different systems always introduces some performance overhead originating from the fact that the data within the memory of a system should be transferred

Serialization	*private static void Serial(Serializable object,* *String filename)* *{* *ObjectOutputStream OS = new ObjectOutputStream(* *new FileOutputStream(filename));* *OS.writeObject(object);* *OS.close();* *}*
Deserialization	*private static Object Deserial(String filename)* *{* *ObjectInputStream IS = new ObjectInputStream(* *new FileInputStream(filename));* *Object obj = IS.readObject();* *IS.close();* *return obj;* *}*

Figure 3.1 – Java pseudo-code of the binary serialization and deserialization

into the memory of the other one. To achieve this, the in-memory object is first transformed into bytes that can be sent over the network, this is called "Serialization". It has two most important reasons:

— the need to persist the state of an object to a storage medium so an exact copy can be re-created at a later stage.

— the need to send the object by value from one application domain to another.

Serialization is the process of converting an object into a form that can be readily transported, which makes it more persistent. For example, an object can be serialized to be easily transported over Internet via HTTP protocol between a client and a server. Deserialization is the reverse process that reconstructs the object from the stream. Since we make use of JAVA language in our conceptions and implementations, it is interesting to know that Java serialization relies on flows. Indeed, the data generated through a serialization process may be in binary format, which we call " *Binary Serialization*", or in a textual format using XML, which is called "*XML Serialization*" [**M.Pich**]. Both types of serialization are described in the following.

3.1.1 Binary Serialization

Binary serialization [**M.Pich**] is the most traditional method for object persistence in Java. It makes use of a pair of objects: *ObjectInputStream* and *ObjectOutputStream*, to read and write a binary format of data. These objects are associated with two classes:*FileInputStream* and *FileOutputStream*, that allow to read and write the bytes to and from a file system. In addition, it is necessary to use the interface "*Serializable*" to identify the mentioned classes and objects. In Figure 3.1, we provide a pseudo-code of the binary serialization and deserialization in Java.

In the serialization process, an object of the *FileOutputStream* class is created, so that an empty file is also created. Then, an instance of the class *ObjectOutputStream* is initialized, such that the content of the concerned data object is transformed into stream of bytes using the method "*writeObject()*". Thus, the objects is the serialized and stored in the empty file. In the deserialization process, a *FileInputStream* object is created to read the serialized content in the

Serialization	*private static void XMLSerial(Serializable object,* *String filename)* *{* *FileOutputStream OS = new FileOutputStream(filename);* *XMLEncoder encoder = new XMLEncoder(OS);* *encoder.writeObject(object);* *encoder.close();* *}*
Deserialization	*private static Object XMLDeserial(String filename)* *{* *FileInputStream IS = new FileInputStream(filename);* *XMLDecoder decoder = new XMLDecoder(IS);* *Object obj = decoder.readObject();* *decoder.close();* *return obj;* *}*

Figure 3.2 – Java pseudo-code of the XML serialization and deserialization

file passed as parameter. This object is used to create an instance of the *ObjectInputStream* class, which allows to load the serialized content into bytes, through the *readObject()* method. These bytes are stored in an ordinary object variable. In both processes, *close()* method closes the file and ends the writing or the reading.

3.1.2 XML Serialization

XML serialization [**M.Pich**] is widely used to ensure the object persistence, particularly in web applications, since it is mainly based on the Java Bean standard that allows the object content to be accessible to other applications, which provides interoperability due to the use of XML standard. This type of serialization makes use of the *FileInputStream* and *FileOutputStream* classes to read and write XML files in the same way as binary serialization. However, for XML serialization the developer has to use the *XMLEncoder* and *XMLDecoder* classes of *java.beans* package to encode and decode the file in XML format. In Figure 3.2, we provide a pseudo-code of the XML serialization and deserialization in Java.

In the serialization process, an object of the *FileOutputStream* class is created, so that an empty XML file is also created. Then, an instance of the class *XMLEncoder* is initialized basing the file stream object, and will be used to encode the data, through the method *writeObject()* that transforms the content of the concerned object into XML to be serialized. In the deserialization process, a *FileInputStream* object is created to read the serialized content in the XML file passed as parameter. This object is then used to create an instance of the *XMLDecoder* class. Afterwards, the XML content of the serialized object is decoded using the *readObject()* method. Finally, the object value is returned without any specific data conversion. In both processes, *close()* method closes the file and ends the writing or the reading.

3.2 XML Serialization with Cryptographic Primitives

A mobile agent must have the ability to communicate with other agents of the system (either local or remote agents), to exchange information and benefit from the knowledge and expertise of other agents. In practice, mobility does not replace the communication capabilities of the agents but completes them. Hence, the interaction between mobile agents needs first to initiate communication between the platforms, ensure their compatibility and collect specific information about them.

To establish a well-defined security policy to address the threats mentioned in the previous Chapter 2, we need to satisfy the security requirements related to this paradigm: authentication, confidentiality, integrity and availability. In this section we present a detailed description of our approach, that consists in simulating a set of cooperative agents in charge of performing different mechanisms, in order to satisfy the security requirements. These mechanisms include the Diffie-Hellman key exchange protocol associated with digital signature to authenticate the communicating entities and ensure integrity of the migrating agent, AES encryption algorithm to guarantee the confidentiality of data exchanged, and the principle of XML serialization to obtain a persistent and transportable format of agent.

3.2.1 Authentication Process

To prevent attacks related to unavailability of authentication, we integrate in each one of the interacting platforms, a specific agent called "$DH - DSA\_Agent$". This later owns a specific list that contains the addresses of the hosts constituting the itinerary to follow. In practice, these addresses can be IP addresses or MAC addresses of the hosting machines. Before the migration of agent to a new host, an authentication mechanism using the Diffie-Hellman key exchange [**DH76**], and the standard for digital signature [**Galla09**] is running between the "$DH - DSA\_Agent$" of both platforms, in order to create a common shared key. This key will be used afterwards to sign and verify the addresses and data exchanged between both hosts.

The first step of Diffie Hellman algorithm is to generate randoms for modulo and primitive root computations. This implies the use of a pseudo random number generator (PRNG) to achieve this task. Yet, the DH-provider in Java Runtime does not support cryptographic generator, which are considered as the faster and most secure ones after the quantum generators. This issue leads us to adopt a new implementation of Diffie Hellman algorithm using ISAAC+ [**Auma06**]. The ISAAC+ algorithm is an enhanced version provably secure of ISAAC (Indirect, Shift, Accumulate, Add and count), which has similarities with RC4 [**Mousa06**]. It uses an array of 256 four-octet integers as the internal state, and writes the results to another 256 four-octet integer array. It is very fast on 32-bit computers.

In our approach, we are inspired from [**Phan05**] in order to fix the weaknesses of the Diffie-Hellman key exchange Protocol, especially its vulnerability to Man-In-The-Middle attack, through integrating the digital signature (DSA). Figure 3.3 enumerates the different steps of the improved protocol. All random values are generated with ISAAC+, the computations are performed on finite field, and for the digital signature we use the one-way function SHA-1 [**East01**]. At the step 10, we introduce the IP address of the remote host (got from the list of addresses that the mobile agent carries) in the signature, while at the step 11, this signature is verified by the hosting platform through producing an equivalent signature using its own IP address.

The main idea of the enhanced integrated Diffie-Hellman-DSA key exchange protocol is to ensure computations basing on two ephemeral secrets v and w, which are chosen by the two

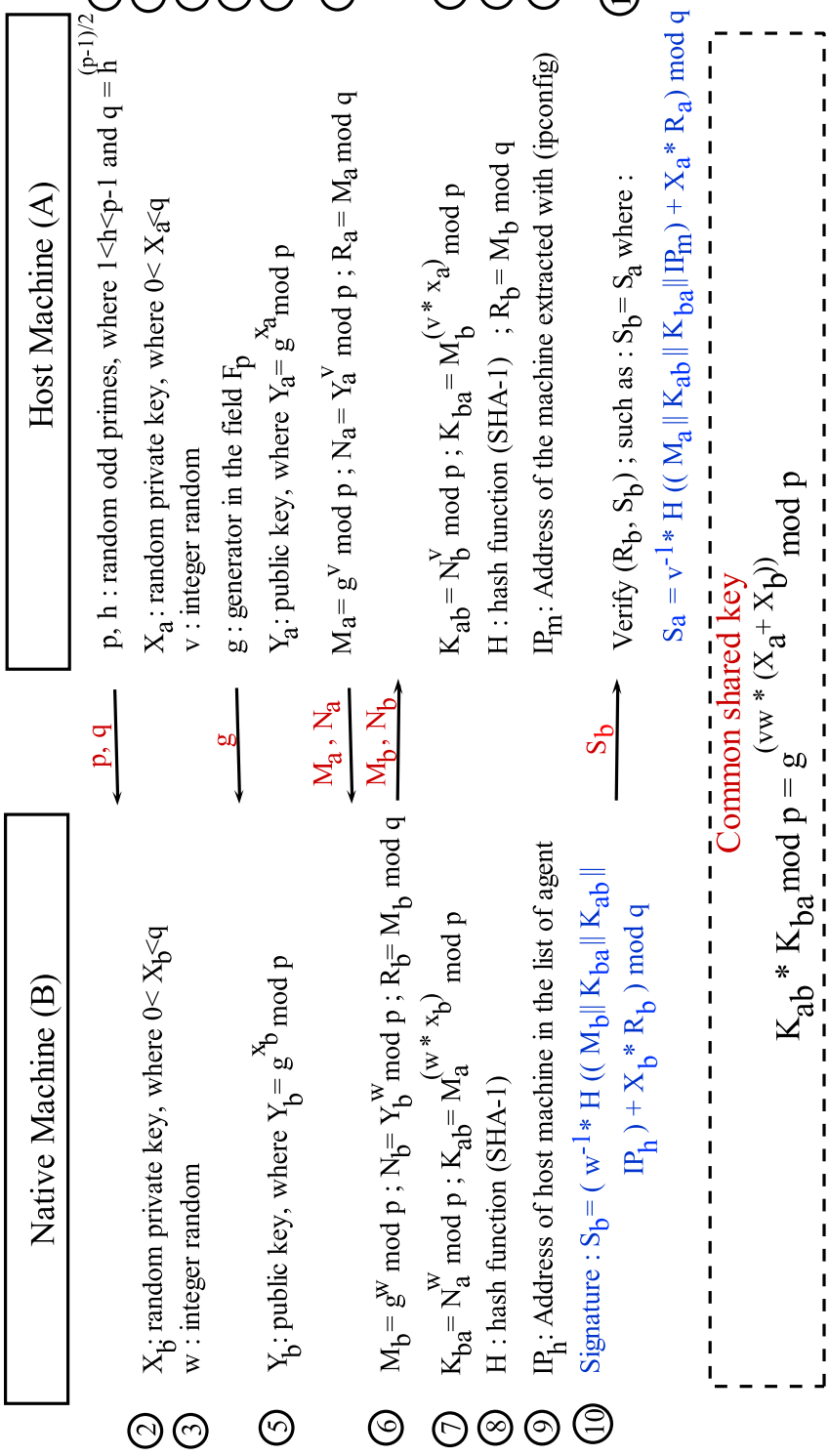

Figure 3.3 – The integrated Diffie-Hellman-DSA key exchange protocol

parties A and B. This provides forward secrecy because even if the long-term private key of any party is exposed, then the previous session keys cannot be computed, since the ephemeral secrets v and w for that session are unknown. In addition, the key freshness property is provided because every session key is a function of ephemeral secrets, so neither party can predetermine a session key's value, since he would not know what the other party's ephemeral secret is going to be.

Figure 3.4 describes the proposed authentication process. First of all, the native machine sends a request for mobility of an agent to the remote platform and asks for information to be able to authenticate it. Since each machine includes a manager agent, which is responsible for managing communications and interactions between the components of its own platform, as well as with the remote ones. Thus, this manager agent collaborates with the "$DH - DSA\_Agent$" in order to perform the steps of the authentication protocol and generate a session shared key of 256 bits. The produced key is used later to maintain the confidentiality and integrity properties.

In fact, the establishment of an authentication mechanism between agents and platforms is very essential, to avoid attacks in relation with unauthorized access. An agent that has access to a platform and its services without having the proper authorization can harm other agents and the platform itself. Thus, a platform that hosts agents must ensure that they cannot get access to the data and services if they do not hold the relevant authorizations. For an efficient and well-defined access control policy, the platform must first authenticate the mobile agent and verify its identity, before being instantiated on the platform. There are several methods for mutual authentication on the network, among them the "HTTPS Mutual Authentication". In this later, secured layers are associated to the protocol using encryption, such as TLS (Transport Layer Secure). Accordingly, secured passwords are generated (TLS-EAP) and the identity of the client and server are verified using certificates. Thus, many attacks are avoided: Man-in-the-Middle attack, offline password dictionary attack and phishing.

However, this mechanism was recently broken as it suffers from serious deficiencies, such as certificates management problem and layering problem, that makes it impossible to match authentication session and transport session. In our solution, we do not make use of certificates, so we gain in terms of resources dedicated to implant a Certificate Authority (CA), and in terms of time consumed to interact with the CA and charge certificates. Besides, in both platforms, there is a distribution of tasks among the agents that are endowed with an execution tracing mechanisms, which creates a matching between different level of execution. Moreover, our protocol is based on complex problems in mathematics such as discrete logarithm.

3.2.2 Confidentiality and Integrity Preserved

When information are exchanged between two environments, especially heterogeneous ones [**Mitro11**], they can easily be intercepted or modified by an intruder. In order to preserve confidentiality in our approach, we are primarily based on the use of cryptography. Indeed, the modern field of cryptography has two major classes: the *symmetric-key cryptography* where the interlocutors share the same key, and the *asymmetric-key cryptography* where each entity holds a pair of mathematically related keys: public key and private key.

At the authentication phase described in the previous section, a session key was created and shared between the two communicating platforms. In addition, the pairs of public and private keys (X_a, Y_a) and (X_b, Y_b), generated respectively by the native and host machines, are used to produce the digital signatures. Hence, the choice to use of one of the mentioned cryptogrpahy classes depends on the conception of our approach, its needs and its performances. Knowing that asymmetric-key cryptosystems are considered to be effective, since they rely on the difficulty

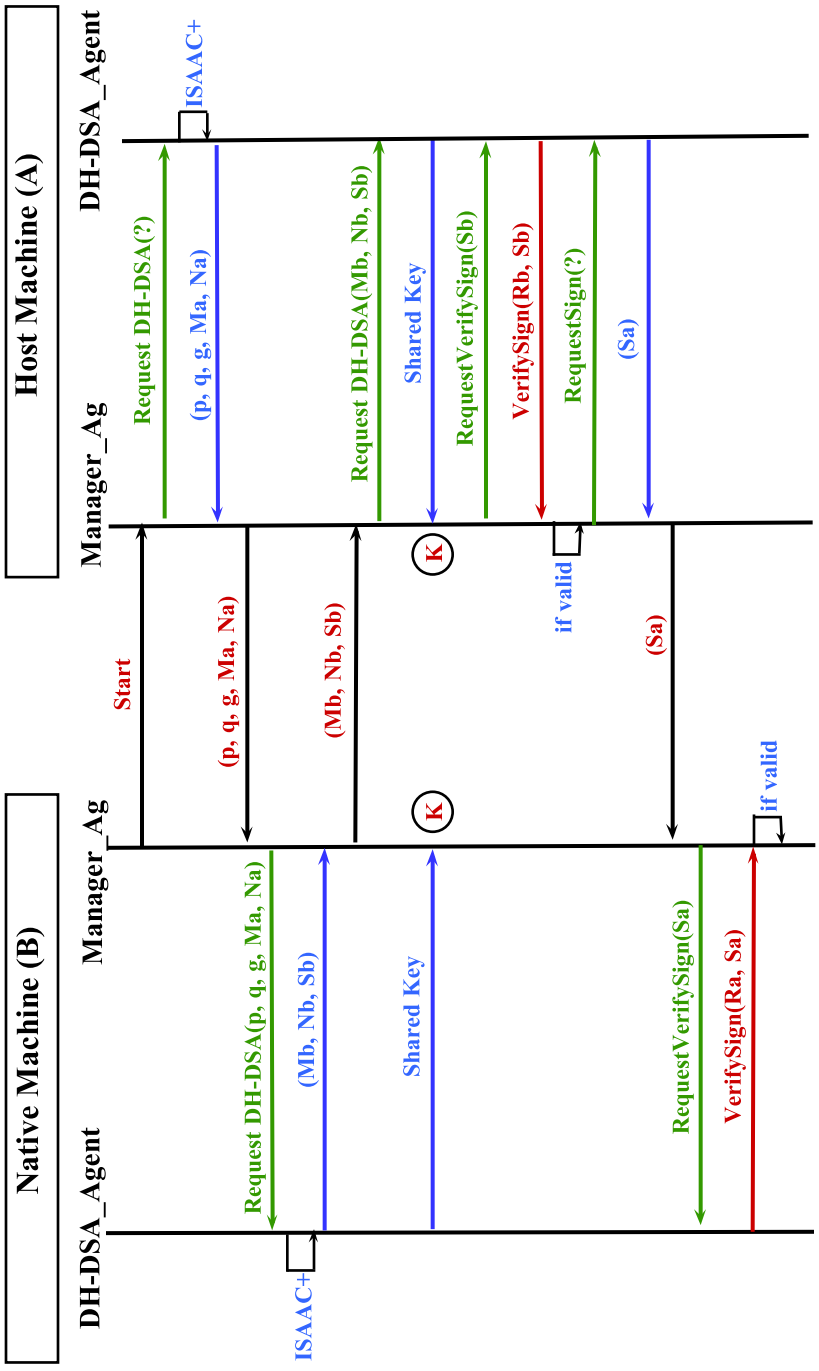

Figure 3.4 – The Authentication process between the Native Platform of the agent and the Remote Platform hosting it

to solve complex problems such as the discrete logarithm. However, they are very slow and consume more computer resources to perform the computations and to store the generated keys, that are mostly very long. On the other side, the symmetric-key cryptosystems make use of less memory and resources with small key length. They are actually much faster in processing because their computations are based essentially on permutations, substitutions and transformations. Besides, they are largely used for data compression, but they need secure channel to share the secret key.

Since the concern of securely exchanging the secret common key is resolved by the authentication process, that adopts a secure key exchange without information leakage. Then, we make the choice to use a symmetric cryptosystem, which is justified by the small length of keys and the fast computations. The algorithm adopted is AES256 (Advanced Encryption Standard) [**Rober12**], which is robust and introduces a key length of 256 bits, that matches well with the length of the obtained key session.

According to this, a specific agent named "AES_Agent" is integrated in the communication, so that it is present in both interacting platforms. This agent involves the cryptographic methods and takes in charge the encryption of the mobile agent on the native machine, including its data and code, while its decryption is assured on the hosting machine. Both encryption and decryption are performed using the session key. It is worth to mention that the "AES_Agent" effectively deals with an instance of the mobile agent class, that is conceived in a persistent format and sent by "$Manager\_Agent$". More details are provided in the next section.

The integrity feature is also preserved in our approach, at two levels: 1) during the authentication-based key exchange, and 2) during the mobility of the agent and exchange of data.

At the last steps of the authentication protocol in Figure 3.4, the native machine, through its "$DH-DSA\_Agent$", signs the set of the generated and calculated values by the protocol, along with the IP address of the remote host. Then, this signature is sent to the relevant host, that in turn charges its "$DH-DSA\_Agent$" to sign the keys generated with its IP address, and sends the signature to the native machine. Both machines must verify and validate the received signature to carry on processing. Once the authentication has passed, the mobile agent is encrypted in order to preserve confidentiality. Before moving the agent to its destination, the same persistent format of the agent instance, used in encryption, is also signed using the session key and the DSA algorithm. The obtained signature is joined to the encrypted agent object. Thus, to check the integrity of the received agent, the hosting platform first decrypts it, signs the given value using the session key and then compares the obtained signature with that provided. If the two signatures are matching, then the agent's integrity is preserved, else the agent was altered.

The unavailability of the integrity may expose the agent to several threats such as "Alteration attack". When an agent migrates to a remote platform, it exposes its own information (the code, the state, and data) to the platform, which can be malicious and may attempt to modify its carried content. Indeed, preventing the agent from being modified by the platform is strongly needed, but unfortunately it does not yet have an efficient solution. Therefore, the impact of the alteration attacks is alleviated through the use of mechanisms able to detect the changes that have occurred.

3.2.3 Mobility Process

Mobility in the context of transportability across the network can be performed through layer protocols, such as HTTP provided with the Apache server. It can also be achieved by

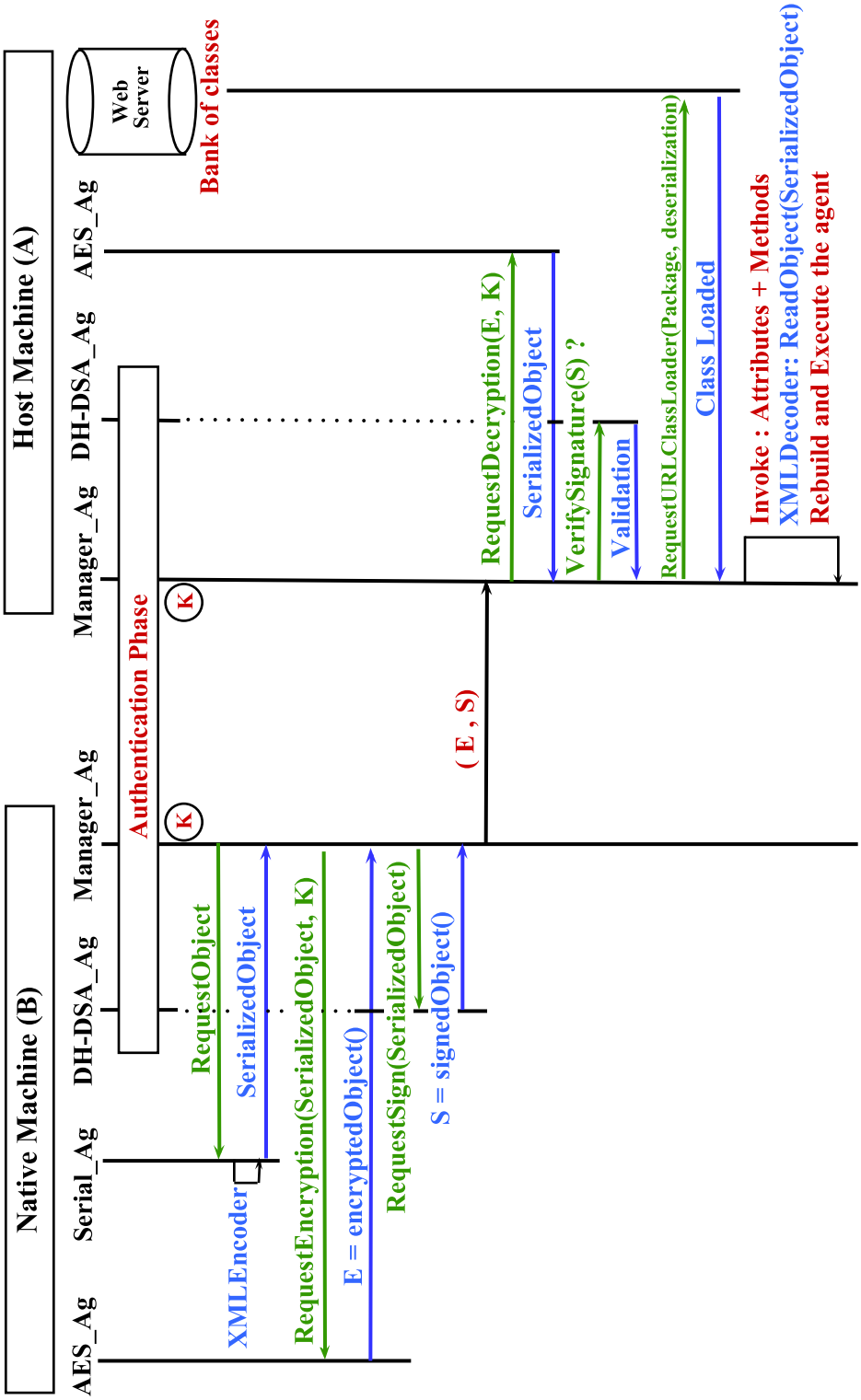

Figure 3.5 – Mobility of the agent as XML serialized object

JAVA APIs like RMI (Remote Method Invocation) or RPC (Remote Procedure Call) that manipulate remote objects. These APIs are structured in network layers based on the OSI model to ensure interoperability between programs and versions of Java.

In our approach, we adopt a weak mobility where the mobile agent restarts its execution on each visited host. For that purpose, we make use of the XML serialization mechanism to provide a persistent and well transportable medium of the agent and its sensible data across the network. This association is appropriate to address the flaws of other modes of transportability, because it is easy to develop and it ensures convenience and efficiency through the generation of easily readable and editable format.

```
AID remoteAG = new AID ("Agent@Hind−47303a:1099/JADE",AID.ISGUID);
        addBehaviour(new CyclicBehaviour (){
            private static final long serialVersionUID = 1L;
            public void action() {
                receive = receive();
                BufferedInputStream bis = new BufferedInputStream(
                            new FileInputStream("C:\\serial.xml"));
                XMLDecoder xmlDecoder = new XMLDecoder(bis);

                Class c = null;
                URL[] urls ={new URL("http://10.9.25.251/")};
                URLClassLoader ucl = new URLClassLoader(urls);
                c = ucl.loadClass("test.serial");

                for (Method m : c.getDeclaredMethods()){
                    System.out.println( " methods : " +m.getName());}
                Class getArg1[] = { (new String[1]).getClass() };
                Method m = null;
                m = c.getMethod("main",getArg1);

                String[] my1 = { "arg1 passed", "arg2 passed" };
                Object myarg1[] = { my1 };
                m.invoke(null,myarg1);
                   ...
                   ... }
```

Figure 3.6 – Java pseudo-code of class loading

Figure 3.5 describes the mobility process using this mechanism. In the native machine, an agent named "*Serial_Agent*" is integrated in order to serialize in XML format an instance (object) of the mobile agent class. This class contains the attributes and execution code of the agent. Then, the XML instance given is encrypted by the "*AES_Agent*" using the session key and transferred to the host machine. At the other side, the class of mobile agent is rebuilt using the encrypted XML instance of the agent, which is firstly decrypted by "*AES_Agent*" using the session key, and deserialized using the XMLDecoder of JavaBeans API.

The problem encountered in this simulation, was when the host platform does not support the JavaBeans of XML serialization or does not recognize certain classes needed in execution.

After research, we have found that the principle of URLClassLoader is the most appropriate for our approach, as illustrated in Figure 3.6. It will give us the possibility to load the classes and packages needed for execution from the URL of native machine or any other databases in the network, which implants an aspect of availability of information.

3.2.4 Evaluation and Results

In this section, we evaluate the performance of our approach through running experiments using JADE agent framework. In the previous section, while giving a description of the solution, we have in parallel analyzed its ability to prevent attacks such as: "Unauthorized Access Attack", "Alteration Attack" and "Eavesdropping Attack". Thus, our evaluation efforts have been focused on testing the time spent by migrating agents, under the conditions mentioned above. This section is divided into two parts: Theoretical Analysis that presents a formal computation of the time spent for interactions, and Practical Experiments in which two tests are performed to calculate the overhead of security provided. The first one is considered as basic test that illustrates a simple migration process without integrating the security mechanisms discussed, while the second one includes the implementation of the proposed approach.

Theoretical Analysis

Normally, a launched mobile agent migrates from one site to another and then comes back to the first one, in what it is called a round-trip. Figure 3.7 shows the stages of the agent round-trip:

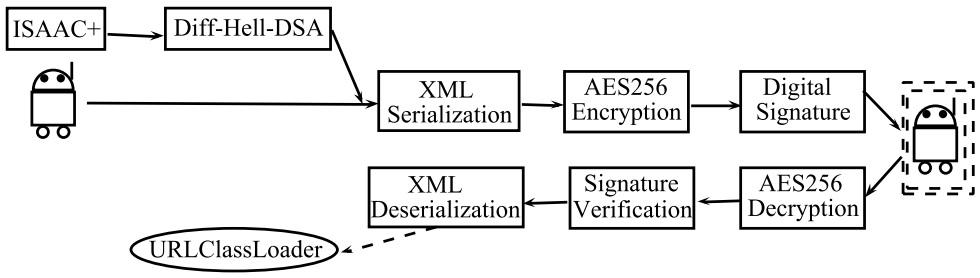

Figure 3.7 – Steps in the round-trip of a mobile agent

— In the going, the agent is involved in an authentication process that uses ISAAC+ and Diffie-Hellman-DSA protocol, and which generates a session key. Afterwards, the agent is serialized in an XML object format, then encrypted using AES256 and signed with DSA.
— Once the serialized-encrypted object is received, the host tries to rebuild the agent, through decrypting the object using AES256 and then deserializing it, while using its signature to check its integrity. When the class of the agent is unreachable, we make use of the URLClassLoader method. Thus, when the agent is rebuilt, an acquittal is sent to the native machine in order to delete the agent, which will be properly executed
— In the return, the agent brings the results of the execution to its native machine.

Let's consider T_{RT} = the time cost of the round trip, such as:

$$T_{RT} = T_1 * N_{jp} \tag{3.1}$$

Where:
- N_{jp} = the number of jumps during the execution, which is based on the data exchanged in each stage.
- T_1 is a period comprising the time costs related to the stages in the proposed approach, as shown in Equation 3.2:

$$T_1 = T_i + T_{dd} + T_{Xs} + T_{enc} + T_{sign} + T_{dec} + T_{Xd} + T_r + T_{mig}. \tag{3.2}$$

such that:
- T_i: the time cost of ISAAC+ randoms generation;
- T_{dd}: the time cost of the Diffie-Hellman-DSA protocol;
- T_{Xs}: the time cost of the XML serialization;
- T_{enc}: the time cost of the AES-256 encryption;
- T_{sign} : the time cost of the DSA signature;
- T_{dec}: the time cost of the AES-256 decryption;
- T_{Xd}: the time cost of the XML deserialization;
- T_r: the time cost of the requests exchanged along the approach;
- T_{mig}: the time cost of the data sending along the agent migration;

Knowing that T_{Xs} is approximately equal to T_{Xd}, and T_{enc} is approximately equal to T_{dec}. Then, the Equation 3.2 becomes:

$$T_1 = T_i + T_{dd} + 2T_{Xs} + 2T_{enc} + T_{sign} + T_r + T_{mig}. \tag{3.3}$$

Since the request messages are internally exchanged within each platform and their length does not exceed few characters, then, no computation time is consumed for the initiation of the communication, no memory is allocated and no frames are used for transporting data across the network. Hence, the time spent for the requests could be considered as negligible, and the Equation 3.3 becomes:

$$T_1 \simeq T_i + T_{dd} + 2T_{Xs} + 2T_{enc} + T_{sign} + T_{mig}. \tag{3.4}$$

As it is previously said, the time consumed during one round trip is calculated according to the Equation 3.1. However, there is an extra time added during the going of the mobile agent. This extra time concerns the loading of the agent's class using URLClassLoader, in case the host does not recognize it. When returning back to the native machine, the mobile agent does not need to load the class because it is already provided. To sum up, the total time spent in a round trip of an agent is calculated according to Equation 3.5:

$$T_{RT} = ((T_1 + T_{cl}) \times N_{Gjp}) + (T_1 \times N_{Rjp}) \tag{3.5}$$

with:
- T_{cl}: the time cost of the class loading;
- N_{Gjp}: the number of the jumps in the go;
- N_{Gjp}: the number of the jumps in the return.

Practical Experiments

Our experiments are performed using 4 heterogeneous machines with different operating systems (Windows, Ubuntu, Mac Os). The first one is considered as a native machine and the

others as hosts. All machines are characterized by Core i7 processor at 2.7 GHz, with 4 Go of RAM, and 500 Go on an ATA 500 hard disk. They are equipped with JADE Snapshot agent platform in its 3.6.1 version, and they make use of a 100Mbps switched Ethernet network with WampServer.

a) The baseline test

In order to evaluate the feasibility and performance of our approach, we find it interesting to configure a baseline test, which consists in performing a simple mobility of agent using only XML serialization concept, without trying to encrypt it or to authenticate it. This allows to calculate the overhead needed for securing the mobile agent. The results of this test are exposed in Table 3.1.

Table 3.1 – Time Performance of the different steps in the baseline test

Time(ms)	**1 Host**	**2 Hosts**	**3 Hosts**
T_{Xs}	**73**	**91**	**143**
T_{mig}	**127**	**243**	**339**
T_{cl}	**338**	**914**	**2463**

In such normal conditions where the agent is only serialized and transferred, Equation 3.4 becomes:

$$T_1 = 2 \times T_{Xs} + T_{mig} \text{ , with } N_{Gjp} = N_{Rjp} = 1 \tag{3.6}$$

Thus, referring to the results in Table 3.1 and according to the Equations 3.5 and 3.6, the time cost of the baseline test, noted T_{RTB}, where an agent moves to one host in normal conditions, can be calculated as follows:

$$T_{RTB} = (((2 \times 73) + 127 + 338) \times 1) + (((2 \times 73) + 127) \times 1) = 611 + 273 = 884ms$$

In theory, the time to move an agent over three hosts is three times the time of moving an agent to one host. However, the use of different and heterogeneous machines for experiments, makes our results not uniform. In order to recognize this overall difference, we compute the average of time that an agent spent while migrating over three hosts:

$$T'_{RTB} = \frac{(((2 \times 143) + 339 + 2463) \times 1)}{3} + \frac{(((2 \times 143) + 339) \times 1)}{3} = 1029 + 208 \simeq 1237ms$$

Then, the difference is around: $\triangle_{RTB} = T'_{RTB} - T_{RTB} = 1237 - 884 = 353ms$, taking into consideration that the large part of this difference concerns the loading of classes over the network, particularly that we have heterogeneous machines.

b) Implementation of our approach

The second experimentation is based on the implementation of our approach, taking into consideration the four aspects mentioned before. This test launches the set of agents charged

with operating the different mechanisms adopted to reach a high level of security. The results we got through running this second test are shown in Table 3.2:

Table 3.2 – Time Performance of the different steps of our proposed approach

Time(ms)	**1 Host**	**2 Hosts**	**3 Hosts**
T_i	**2.4**	**3.9**	**6.1**
T_{dd}	**3.8 (parameters) + 2.6 (computations) = 6.4**	**20.8**	**48**
T_{Xs}	**78**	**96**	**141**
T_{enc}	**15**	**33**	**50**
T_{sign}	**10.4**	**22.7**	**34**
T_{mig}	**182**	**328**	**432**
T_{cl}	**343**	**907**	**2470**

Referring to these results and according to the Equations 3.4 and 3.5, with: $N_{Gjp} = N_{Rjp} = 1$, the time spent by our approach to move an agent is noted T_{RTA}, and can be calculated as follows:

$$\begin{aligned} T_{RTA} &= ((2.4 + 6.4 + (2 \times 78) + (2 \times 15) + 10.4 + 343 + 182) \times 1) + \\ &\quad (((2.4 + 6.4 + (2 \times 78) + (2 \times 15) + 10.4 + 182) \times 1) \\ &= 730.2 + 387.2 \\ &= 1117.4ms \end{aligned}$$

Similarly to the baseline test, we are interested in computing the time difference between the agent migration to one host and the average of agent migration to three hosts :

$$\begin{aligned} T'_{RTA} &= \frac{((6.1 + 48 + (2 \times 141) + (2 \times 50) + 34 + 2470 + 432)) \times 1}{3} + \\ &\quad \frac{(((6.1 + 48 + (2 \times 141) + (2 \times 50) + 34 + 432) \times 1)}{3} \\ &\simeq 1124 + 300.7 \\ &\simeq 1424.7ms \end{aligned}$$

Then, the difference is around:

$$\triangle_{RTA} = T'_{RTA} - T_{RTA} \simeq 1424.7 - 1117.4 \simeq 307.3ms.$$

Finally, we can deduct the overhead dedicated to secure the mobility of the agent across the network. This overhead is mainly related to the security techniques employed to ensure the authentication between the communicating entities, as well as to preserve the integrity and confidentiality of data and exchanges. The computational value of this overhead is:

$$T_{RTA} - T_{RTB} = 1117.4 - 884 \simeq 233ms.$$

This value represents 21% of the overall time that an agent takes to move. It involves the cost of: ISAAC+ randoms generation, the key exchange, the DSA digital signatures, the AES encryption/decryption, the loading of agents classes using URLClassLoading and the exchange data between hosts and agents. The overhead of $233ms$ seems to be admissible, credible and not compromising the mobility performance of the agent, which benefits from a security feature protecting him against vulnerabilities of hosting platforms.

3.3 Binary Serialization with ID-based Key Agreement Protocol

The mobility of the agent and its capacity to roam freely over network nodes, does not substitute the fact that it must hold the ability to well communicate with its entourage, in order to exchange information and benefit from the knowledge and expertise. Thus, it is quite important to initiate the communication between the interacting entities, check their compatibility and authenticate them, before exchanging sensible data end executing tasks.

This section provides a deep description of a new approach to secure the mobile agent, that makes a set of agents interacting and cooperating together, in order to satisfy the security requirements of the system. This approach includes diverse mechanisms, such as ID-Based Key Agreement protocol to authenticate the interacting entities and to generate a session key, which is used afterwards in AES encryption to ensure the confidentiality of the exchanged data along. A binary serialization of the mobile agent is integrated to guarantee a persistent and easy transportable format over the network, while a Schnorr signature is used to preserve its integrity.

First of all, we find it interesting to make an overview of the agents architecture adopted to simulate the proposed approach. This architecture is illustrated in Figure 3.8, and it integrates a set of cooperative agents in the two interacting platforms:

— "**Admin_Ag**" is the administrator agent charged with managing the communication and data exchange between the two platforms. Each incoming/outgoing request or information to/from the platform is controlled and analyzed by this agent. It has also to administer the communication between the local agents and direct the tasks associated with them.
— "**IKA_Ag**" is the agent charged with performing the ID-based key agreement protocol and the authentication.
— "**BinSerial_Ag**" is the agent charged with carrying out the binary serialization and deserialization of the mobile agent object.
— "**AES256_Ag**" is the agent charged with encryption and decryption of the mobile agent, using the session key generated during the authentication.
— "**Sign_Ag**" which is responsible for editing a Schnorr signature of exchanged data, in order to check their integrity.

3.3.1 Authentication Process

Authentication is an important process that verifies the identity of an entity (person, computer...), in order to decide whether to grant him access to the resources (systems, networks, applications...) or not. Thus, an authentication process between the native (initiator) platform

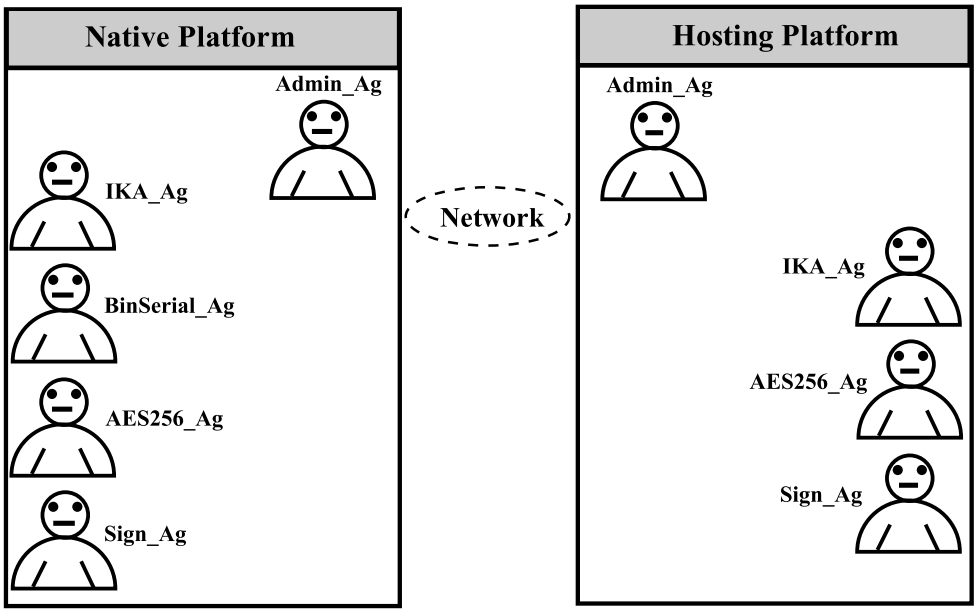

Figure 3.8 – The Agents Architecture of the Communicating Platforms

of the agent and the platform receiving it, is the first and essential resource before the execution of the agent. This is due to the fact that a hosting platform could be malicious, and may hide its real identity in order to attract the agent and then leads attacks on it.

In our approach, the authentication process is operated by a specific agent called "IKE_Ag", which is integrated in both communicating platforms. This agent achieves an ID-Based Key Agreement protocol (IKA) [**Can02**], that allows both to authenticate the platforms and to generate a secret session key, which will be used to encrypt the information exchanges. This protocol assumes that the involved parties are identified by a unique and secret identity (e.g., Alice's identity is a string value (ID_A) that may include the hash of various concatenated parameters, such as: JADE identifier with username, IP address, MAC address and realm in randomized order). Moreover, a trusted entity called Key Generation Center (KGC) is implanted to generate the public parameters of the system, and also to issue the secret keys to the users, based on their public identities. Our proposed IKA protocol is inspired from [**Fiore10**] and improved to fit our specific needs and conditions, as well as to ensure the targeted authentication. This protocol is based on three phases: Setup, Key Extraction and Key Exchange.

It is worth to mention that for simple and deterministic simulation, we consider that the "*IKA_Ag*" contains a beforehand list of the platforms' identifiers that it has the intention to visit. This list is retrieved from a Trusted Third Party (TTP), and it respects the confidence threshold and the constraints imposed by the native platform. This concept can be applied similarly for the dynamic and adaptive mobility, in which the agent ignores where it has to migrate and takes a decision once it found attractive indexes in real-time.

Figure 3.9 describes the process of the improved IKA-Authentication protocol. In the setup phase, public parameters to be used in computations and exchanges are generated by the KGC, such that the random numbers are generated using ISAAC+ random generator [**Auma06**], and the operations are performed on a finite field. The protocol effectively begins when the native

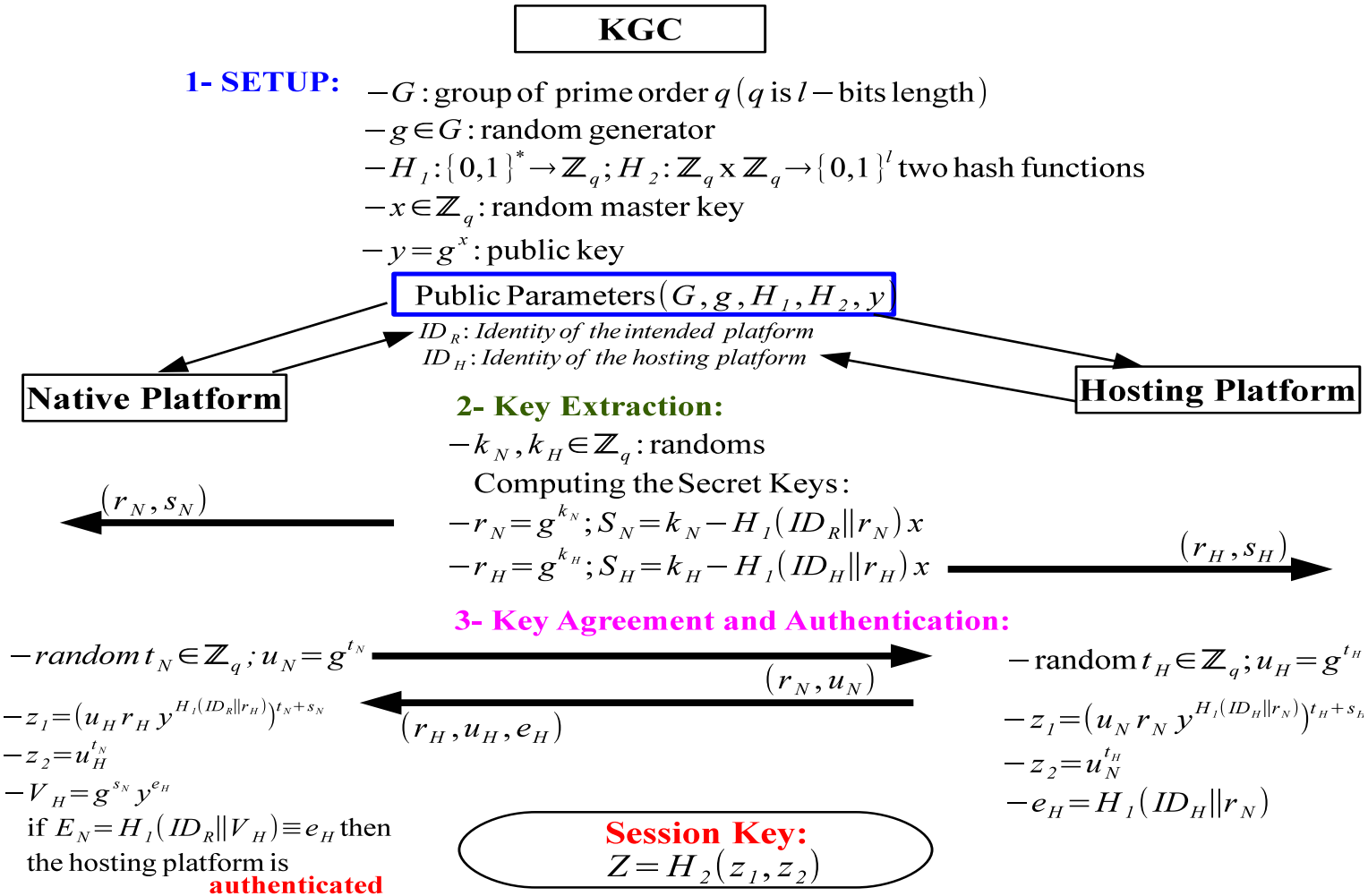

Figure 3.9 – The integrated improved ID-based key agreement protocol

platform sends to the KGC the identity of the hosting platform, that the mobile agent wants to authenticate before migrating to it. Once informed by the request from the KGC, the hosting platform forwards its own identity to be verified and authenticated. At the extraction phase, a private key is computed and forwarded to each one of the interacting platforms by the KGC. This key is based on the Schnorr's Signature [**Schn91**] of the platform identity using the public key parameter y. Afterwards, at the key exchange phase, both platforms exchange credentials based on random exponents. Among the credentials sent by the hosting platform, there is e_H: the hash value of its own identity ID_H concatenated with the public r_N. Therefore, In order to verify the identity of the remote host, the native platform computes the hash of the intended platform identity ID_R concatenated with the value of V_H, and checks if it corresponds to the value of e_H. Finally the session key is computed as follow:

$$Z = H_2(z_1, z_2) = H_2(g^{(t_H+S_H)(t_R+S_R)} g^{t_H.t_R})$$

Figure 3.10 illustrates the authentication process and the interactions between the agents of the two communicating platforms. First, the "*Admin_Ag*" of the native platform notifies the hosting platform with its intention to move an agent, and advises it of the necessity to be authenticated. Then, an authentication process is launched, integrating the "*IKA_Agents*" of both platforms and the KGC. At the end, if the authentication succeeds, then a session key of 256 bits is generated to be used later to maintain the confidentiality and integrity of data exchanged.

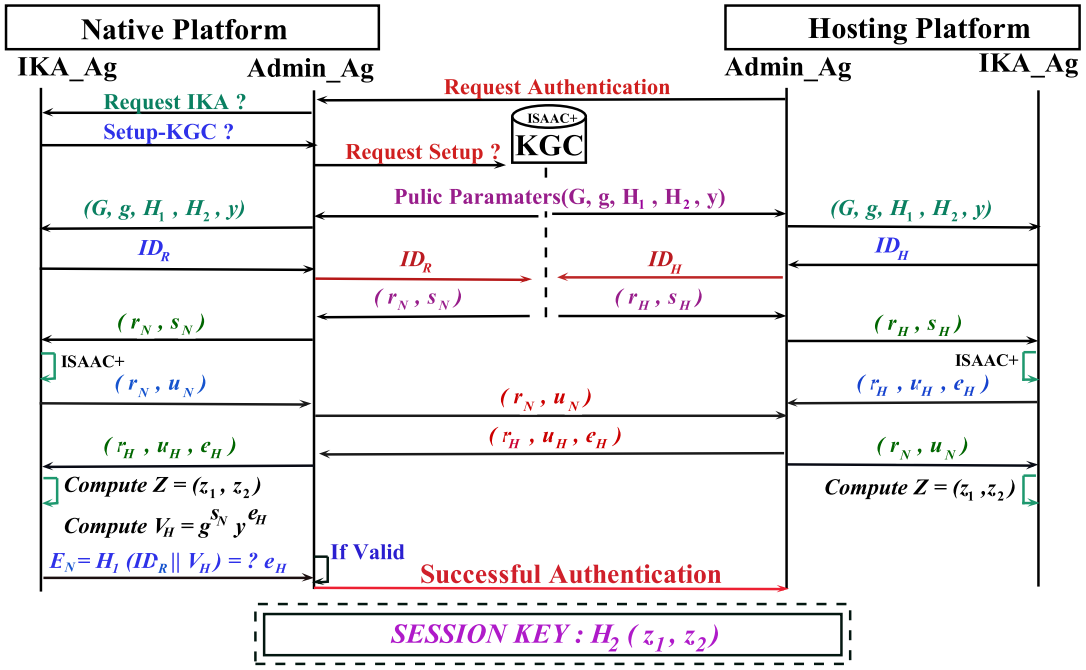

Figure 3.10 – The authentication process between native platform and hosting platform

In general, the authentication is an essential mechanism that avoids attacks based especially on unauthorized access. Definitely, there are several protocols and functions to ensure

authentication, such as protocols that combine TLS (Transport Layer Secure) with Certificates Authority (CA). However, this mechanisms are easily broken regarding the topological architecture of the network, that makes it impossible to match authentication session with the transport session, and where the management of certificates, in terms of space allocation and time, is arduous. In this context, our proposed authentication based on IKA protocol is proven to be secure, since it depends on the new intractability assumption called *Strong Diffie-Hellman assumption* (SDH), and provides many security properties [**Fiore10**]:

— **Forward Secrecy**: after that a session is completed and that its session key is erased, the adversary cannot learn anything about this key even if it corrupts the parties involved in that session.
— **Resistance to Reflection Attacks**: an adversary cannot compromise a session in which the two parties have the same identity (and the same private key).
— **Resistance to Key Compromise Impersonation**: when the adversary knows the private key of a platform, it does not allow him to impersonate this platform to other entities.

3.3.2 Confidentiality and Integrity Preserved

Confidentiality ensures that information are kept secret and that only authorized users can get access to them. It is a very recommended property in the conception of security policies. Thus, maintaining the confidentiality of information circulating through the network is a big concern, either for the resources and services of the hosting platform or for the code and data of the mobile agent.

Our contribution to fix this issue is primarily based on cryptography, taking into consideration its two main categories: symmetric-key cryptography and asymmetric-key cryptography, with their advantages and limitations. In the authentication phase, the ID-Based Key Agreement protocol allows the share of a session key of 256 bits in a secure manner. This provides a solution to the famous problem of securing the key exchange in symmetric-key cryptography. Hence, we think it very interesting to employ this category along with the session key, especially for its high performances with the data compression and with the devices having limited storage memory, such a mobile agent. Accordingly, we make use of the Advanced Encryption Standard (AES) [**Rober12**], that is qualified as a robust encryption algorithm, introducing different key lengths (128, 196 or 256 bits).

After producing a persistent object of the mobile agent class, the "*Admin_Ag*" sends this object along with the session key to the "*AES256_Ag*". This later encrypts the persistent object and returns back the cipher to the "*Admin_Ag*", that dispatches it in a migration process. Through encrypting the data exchanged between the authenticated platforms, we counter the eavesdropping attacks and avoid that intruders recognize the real content of the exchanged messages that are intercepted.

Concerning the integrity, it ensures that the data are those believed to be. This implies the verification that the data have not been altered during the transmission (either accidentally or intentionally). In order to maintain the confidence between the communicating parties of our approach, the integrity of the exchanged data is preserved such that each platform makes use of Schnorr's signature [**Schn91**] to sign its own data, before sending them to the intended destination. The Schnorr' signing and verifying is described in Figure 3.11.

At the authentication phase, the hosting platform signs its identity along with r_N (the signature corresponds then to the pair (S_N, e_H)), while the native platform has to verify this signature to ensure authentication, as well as to prove integrity. The appropriate computation

Figure 3.11 – The signing and verifying of Schnorr' signature

Signing	**Verifying**
- x: the master key of the system - $k \in Z$: random To sign the message M: 1) $r = g^k$ 2) $e = H(M \parallel r)$ 3) $S = k - ex$ The signature is the pair (S,e)	- Y: the public key of the system 1) $r' = g^S.Y^e$ 2) $e' = H(M \parallel r')$ if $e == e'$ then the signature is verified

of the Schnorr' signature is shown in the last steps of Figure 3.10.

Once the authentication is successful, the mobile agent could be welcomed by the hosting platform. In reality, this latter receives an encryption of the agent's instance along with its corresponding signature. Then, the platform begins by decrypting the encrypted agent's instance, signing the given result using Schnorr' signature and then comparing the obtained signature with that provided with the cipher. If both signatures are matching, then the integrity of the mobile agent received is proven. This implies that the hosting platform can proceed to the rebuild of the mobile agent's class with all its attributes and functions, and then launch its execution. However, if the signatures are not corresponding, then the hosting platform terminates the mobile agent and sends a report to the native platform, in order to inform it about the failure of integrity check.

The use of the signature mechanism, to support the integrity of information communicated between two platforms, avoids several threats such as "Alteration Attacks" and "Pollution Attacks". However, being able to prevent an agent from being modified by a platform or at least detect the changes being occurred, are problems which still do not find a radical solution.

3.3.3 Mobility Process

The systems based on mobile agents support two types of mobility: weak and strong. In the weak mobility, only the code, data and some special moments (breakpoint) are transferred with the agent to be run. It may be initiated at any time but with loss of current treatment, i.e., it resumes its execution from the beginning. The systems which support weak mobility, save and restore only the runtime values of (all or some of) the agent's properties, while the agent's execution is resumed on the target machine by, for example, automatically sending a predefined message to the agent. On the other hand, the strong migration allows an agent to move across the network, regardless of the runtime state in which it is located, and to continue its execution after migration exactly where it was stopped before. It includes saving and restoring the entire runtime state of the agent, including the execution stack, the program counter, and so on. Strong mobility is more transparent, in the sense that agent is completely unaware of it. However, it is technically more difficult to implement.

In practice, the mobility of agents can be performed using data transfer protocols based on TCP/IP, such as: HTTP, FTP, SMTP and RTP. These protocols are usually connected with a networking service such as Apache Server, where data are exchanged as sockets. Another possibility to achieve the mobility is to make use of JAVA APIs like RMI (Remote Method Invocation) and RPC (Remote Procedure Call) to monitor distant entities.

In our approach, we adopt a weak mobility in the context of agent transportability across the network. Moreover, to enhance this transportability to make it persistent and easier, it

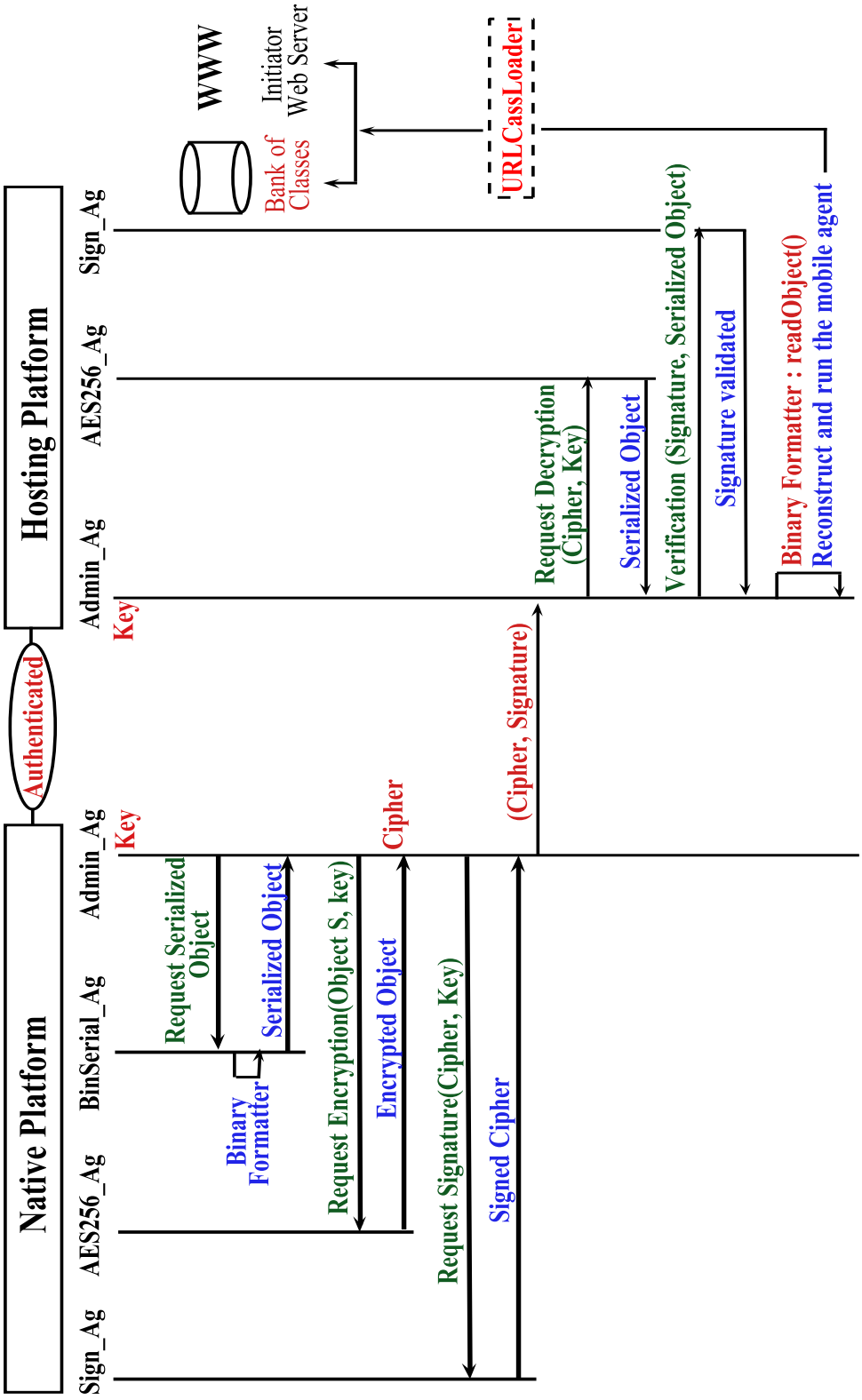

Figure 3.12 – Mobility of the agent as binary serialized object

appears very interesting to make use of a binary serialization mechanism. This later presents many advantages since it deals with several issues of other transfer modes, generates a readable and editable format which provides efficiency and it is easy to implement.

When the hosting platform is successfully authenticated, the native machine prepares its agent to move. For that purpose, an agent named "*BinSerial_Ag*" is integrated, which takes in charge to serialize the instances of the mobile agent class (object), using Serializable and Externalizable interfaces along with a set of writing and reading methods on output and input streams. In this process, all attributes and variables of the mobile agent as well as its code are coded as a binary data flow. According to **[Tau12]**, the JAVA language provides tools (in JDK) that allow the binary serialization in transparent way and independently of the operating system.

Figure 3.12 shows where and when the process of binary serialization is introduced in the mobility of the agent. Before transferring the serialized instance of the mobile agent to the remote hosting platform, it is firstly encrypted using the session key generated in the authentication phase. Once arriving to the hosting platform, the mobile agent instance is decrypted and then deserialized using the BinaryFormatter of the JAVA serialization package. This aims to reconstruct the class of the serialized object on the intended platform. If the skeleton of the mobile agent class is not known by the API of the JAVA core, then the hosting platform may call the dynamic class loading services provided by JAVA such as *URLClassLoader*, which load the predefined class from a bank of class provided on the network, or from the web server of the native platform after obtaining the necessary authorizations.

3.3.4 Evaluation and Results

In this section, we expose and discuss the results got through implementing the proposed solution. For that purpose we make use of JADE **[Belli01]**, the Java Agent Platform designed for multi-agent systems and with respect to specifications of FIPA **[FIPA02]** standard specified for software interoperability among agents and agent- based applications. First, we give a theoretical computation of the time spent to achieve all aspects of the solution. Then, we present the results of the testing performed to know the overhead provided through introducing security mechanisms.

Theoretical Analysis

During the trip of the agent from the native platform to the hosting platform, it performs many operations to ensure the different aspects of security needed. According to this, Figure 3.13 illustrates the round-trip of the mobile agent, that includes:

— In the going, the agent is involved in an authentication process that uses ISAAC+ and an ID-based Agreement protocol (IKA), that eventually produces a session key. Subsequently, the agent is serialized in a binary object format, and to preserve confidentiality and integrity, the serialized object is encrypted using AES256 and signed using Schnorr' signature.

— Once the encrypted object is received, the host tries to rebuild the agent, through decrypting the object using AES256 first, checking its integrity using signatures and then deserializing it. When the class of the agent is unreachable, we make use of the URL-ClassLoader method. Thus, when the agent is rebuilt, an acquittal is sent to the native machine in order to delete the agent, in order to be properly executed.

— The agent returns back to its native machine and presents the results of the execution.

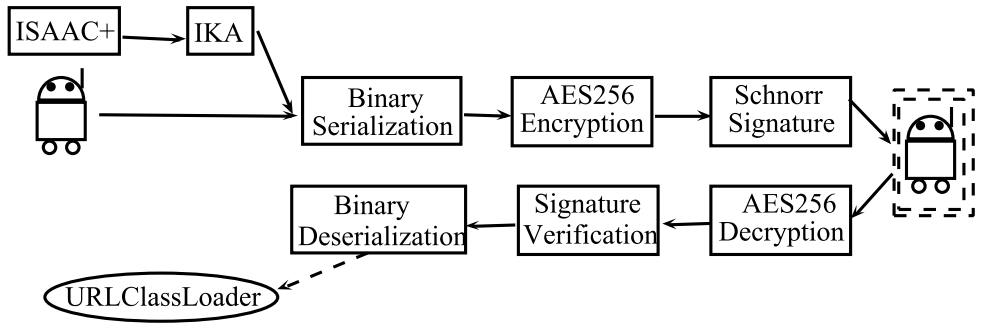

Figure 3.13 – Steps in the round-trip of a mobile agent

Let us consider T_{RT} the total time spent for the approach, according to Equation 3.1 and 3.5. Thus, the included period T_1 comprises many processes represented by the sub-durations in Equation 3.7:

$$T_1 = T_r + T_i + T_{IKA} + T_{Bs} + T_{enc} + T_{sign} + T_{mig} + T_{dec} + T_v + T_{Bd}. \tag{3.7}$$

such that:

— T_r: the time cost of the requests exchanged along the approach;
— T_i: the time cost of ISAAC+ randoms generation;
— T_{IKA}: the time cost of the ID-based key agreement protocol;
— T_{Bs}: the time cost of the binary serialization;
— T_{enc}: the time cost of the AES-256 encryption;
— T_{sign} : the time cost of the Schnorr signature;
— T_{mig}: the time cost of the data sending along the agent migration;
— T_{dec}: the time cost of the AES-256 decryption;
— T_v: the time cost for the verification;
— T_{Bd}: the time cost of the binary deserialization;

In order to simplify the computations, we will take into consideration some constraints. Knowing that the messages transporting requests are small and they are exchanged internally, so T_r can be negligible. Also, T_{enc} is estimated to be approximately equal to T_{dec}, as well as T_{Bs} with T_{Bd}. Then, Equation 3.7 becomes:

$$T_1 = T_i + T_{IKA} + 2T_{Bs} + 2T_{enc} + T_{sign} + T_{mig} + T_v. \tag{3.8}$$

Practical Experiments

The practical testing of the proposed approach is conducted using one machine as a Native Platform and four others considered as Hosting Platforms. All machines are heterogeneous with different operating systems (2 Windows, 2 Ubuntu, and 1 MacOs). The technical characteristics of the five machines are: Intel Core i7 processor at 2.7 GHz with 4 Go of RAM, connected through 100Mbps switched Ethernet network with WampServer [**WampServ**]. For the creation, management, mobility and execution of the agents we adopt JADE Snapshot of version 4.3. For the IKA-Authentication protocol, SHA-2[**Gilb03**] and SHA-3[**Jaffar13**] are used to generate

a hash of 256 bits. Table 3.3 shows the timing results obtained during the trip of the mobile agent and the execution of different security processes.

Table 3.3 – Time Performance of the different steps of our proposed approach

Time in ms	**1 Host**	**2 Hosts**	**3 Hosts**	**4 Hosts**
T_i	2,9	5,6	8,3	11,4
T_{IKA}	8,7	16,4	32,3	51,1
T_{Bs}	59,5	92,2	148,7	182,6
T_{enc}	14	30,6	48,4	82,8
T_{sign}	12,7	24	33,9	45,6
T_{mig}	179,4	327,2	496,4	682,7
T_v	2,7	5,2	7,8	10,2
T_{cl}	313	882	1863	2911

Thus, according to Equations 3.5 and 3.8 with $N_{Gjp} = N_{Rjp} = 1$, and referring to the obtained results in Table 3.3, the total time spent to execute a one round-trip of our approach, noted T_{RTA}, is:

$$\begin{aligned} T_{RTA} &= ((2,9+8,7+(2\times 59,5)+(2\times 14)+12,7+179,4+2,7+313)\times 1)+ \\ &\qquad ((2,9+8,7+(2\times 59,5)+(2\times 14)+12,7+179,4+2,7)\times 1) \\ &= (340,7+343)+343 \\ &\simeq 1026,7ms \end{aligned}$$

In the logic of the calculations theory, the time that takes the agent to move over four hosting platforms, must be four times equal to the time of agent's migration to one hosting platform. We think it interesting to see if our solution and the architecture chosen respect this theory settling. Thus, to know the overall difference, we calculate the total time to execute our solution for 4 hosting platforms, noted $T^{'}_{RTA}$:

$$\begin{aligned} T^{'}_{RTA} &= \frac{(11,4+51,1+(2\times 182,6)+(2\times 82,8)+45,6+682,7+10,2+2911)}{4}+ \\ &\qquad \frac{(11,4+51,1+(2\times 182,6)+(2\times 82,8)+45,6+682,7+10,2)}{4} \\ &= 1060,6+332,9 \\ &\simeq 1393,5ms \end{aligned}$$

Therefore, the overall difference is:

$$\triangle_{RTA} = T^{'}_{RTA} - T_{RTA} = 1393,5 - 1026,7 = 366,8ms$$

Taking into consideration that the large part of this difference concerns the loading of classes over the network, we can conclude that our approach scales regarding to the changing of environments and the increase of visited hosts. This is mainly due to the dynamic and flexible

aspects of the agent's behaviors, as well as to the use of JAVA which allows us to benefit from its tracing services to store traces of execution code in cash memory.

Finally, from above experiments we can deduce the overhead of the security added to the mobility of the agent. This overhead is strongly related to the mechanisms employed in view of avoiding threats raising due to the unavailability of authentication, integrity and confidentiality. Hence, it includes the cost of ISAAC+ random number generator, IKA-Authentication protocol, AES encryption and Schnorr signature with its verification. For that purpose, let consider a baseline test considering the following measurements In Table 3.4:

Table 3.4 – Time costs of the operations in the baseline test

Time(ms)	**1 Host**	**2 Hosts**	**3 Hosts**
T_{Bs}	**61**	**84**	**129**
T_{mig}	**92**	**211**	**323**
T_{cl}	**308**	**870**	**1814**

Thus, referring to the results in Table 3.1 and according to the Equation 3.5, such that :

$$T_1 = 2 \times T_{Bs} + T_{mig} \text{ , with } N_{Gjp} = N_{Rjp} = 1 \tag{3.9}$$

Then, the cost of an agent mobility to one host, in normal conditions, is noted T_{RTB} and can be calculated as follows:

$$T_{RTB} = (((2 \times 61) + 92 + 308) \times 1) + (((2 \times 61) + 92) \times 1) = 522 + 214 = 736ms$$

Finally, the computational value of the security overhead is:

$$T_{RTA} - T_{RTB} = 1026,7 - 736 \simeq 290,7ms.$$

This security overhead represents approximately 28% of the total time spent to move the agent in secure manner, using the proposed approach. In practice, this value seems to be eligible and credible, and did not affect the flexibility of the mobility aspect, which in reality gains in matter of protecting him from any external threats.

3.4 Synthesis and Discussion

In this section, we conduct a comparison between the two proposed approach. According to this, many evaluation performances are considered in terms of energy consumption, time cost and security overhead.

Figure 3.14 illustrates the energy consumption rate of the first approach using XML serialization compared to the second approach based on binary serialization. We notice that the first approach expresses more consumption of energy than the second approach. This expands with the increase of the visited hosts.

In addition, Figure 3.15 presents a comparison of the time performance between the two proposed approaches, where it is clearly shown that the second approach needs less time costs compared to the first one. This could be due to the binary context of the agent, which fastens its migration and processing since it is easily recognized.

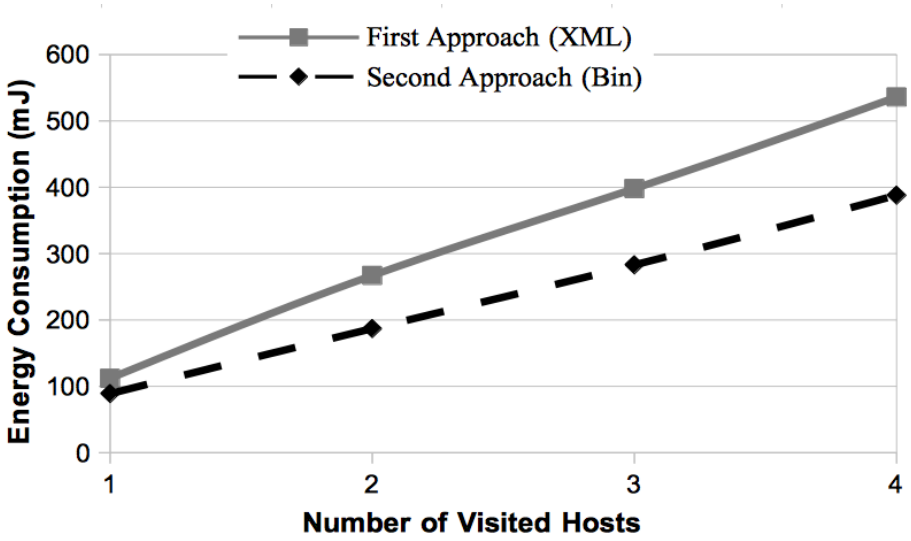

Figure 3.14 – Comparison of energy consumption between the two proposed approaches

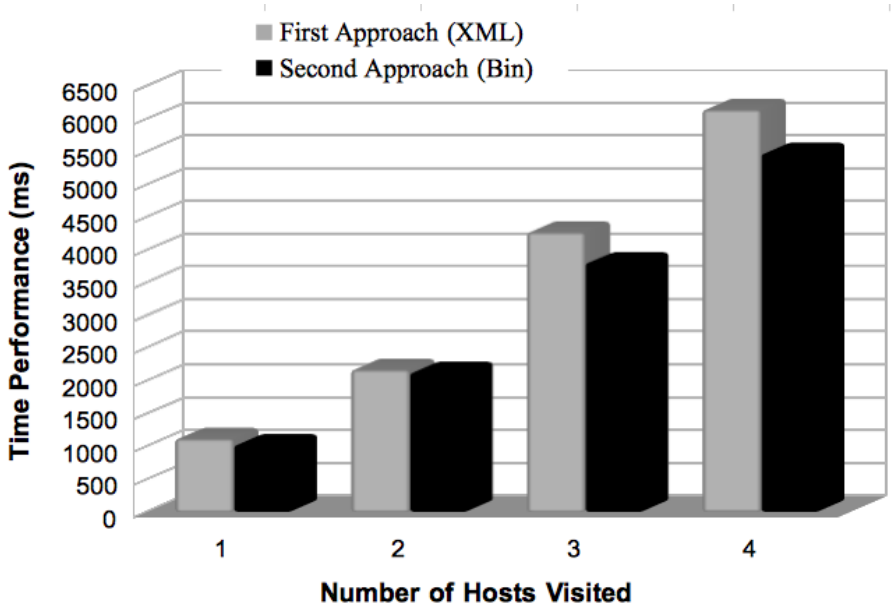

Figure 3.15 – Comparison of time performance between the two proposed approaches

Figure 3.16 evaluates the security overhead of the two proposed approaches. Since both approaches address the same security aspect, we can spot that the first approach base on XML serialization consumes less overhead to maintain the required security compared to the second approach based on binary serialization.

Finally, we can deduce that each one of the proposed approaches holds specific benefits

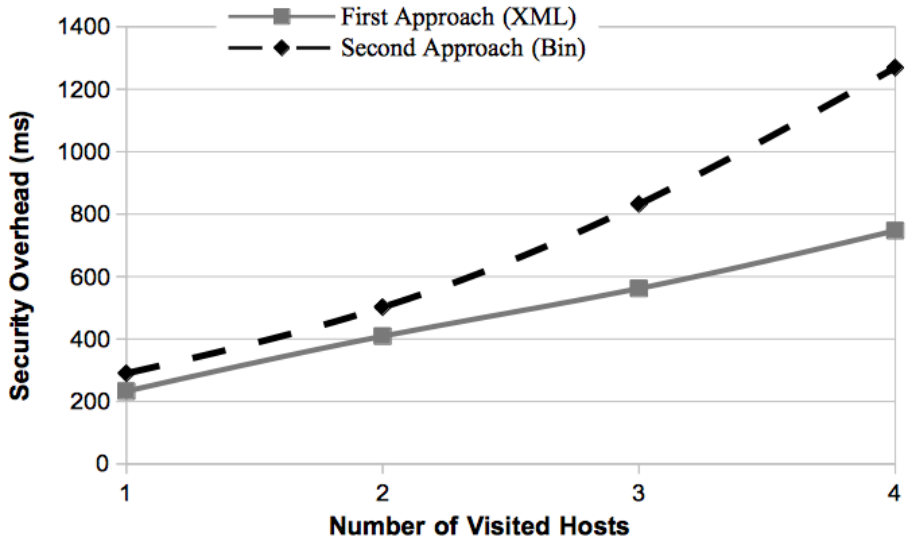

Figure 3.16 – Comparison of security overhead between the two proposed approaches

and limitations. Despite that the first approach consumes more energy and time, it shows a low security overhead. Thus, it can be appropriate for highly sensible applications where security is a priority. In parallel, the second approach expresses lower rates of energy and time consuming face to a high security overhead, which makes it adequate for applications where security is strongly needed, as long as it does not compromise the other features like flexibility and reliability.

3.5 Conclusion

This chapter presents initial results of a research effort aimed at the analysis of the security issues in mobile agent systems. It also describes two proposed solutions to elaborate security policies based on mechanisms such as: cryptography, signature, key exchange and key agreement protocols, as well as XML and binary serialization. These mechanisms are mainly introduced to ensure the properties of confidentiality, integrity, authentication, and give an enhanced mobility for the agent migrating from one site to another across the network. The results of the mobile agent performance using these approaches, and the security analysis provided by evaluating their effectiveness to prevent attacks such as: "Man-in-the-middle attacks", "Masquerading attacks", "Eavesdropping attacks" and "Alteration attacks", can be reused as guidelines to develop secure mobile agent systems involved in mission-critical applications.

Chapter 4

Elliptic Curve Cryptography for Mobile Agents

Contents

In this chapter, a novel approach is presented to early identify anomalies and intrusions occurred during the trip of the mobile agent. This approach relies on two mechanisms: the anonymous authentication and the intrusion detection. The proposed authentication scheme is provided with a privacy preservation and an anonymity in the mobility across the networks. It is supported by the use of elliptic curve cryptography and bilinear pairing, that guarantee reliability and efficiency in countering famous and vicious attacks. In addition, our agent is endowed with an intrusion detection system based on cryptographic traces, where records are produced during the execution of requested tasks on hosting platforms, with respect to a chaining mechanism. This latter substantially reduces the size of the nested and stored traces and makes a connection among all of them until return to the native platform, which facilitates their verification, as well as the revocation of the related malicious hosts.

The rest of this chapter is organized as follows. In Section 2, some notations used in this paper with a highlight on the employed mechanisms are provided. Section 3 presents the process of the anonymous authentication based on elliptic curve cryptography, such that a session key

is produced through an associated key agreement to encrypt and decrypt the exchanged messages among the authenticated entities. In Section 4 our proposed intrusion detection method is described as mainly based on tracking the behaviors of the agent, during its trip across multiple platforms. Thus, it keeps a proof of the agent execution or alteration, through producing traces that employ cryptographic primitives and respect a chaining mechanism. Besides, several evaluations based on security analysis and performance analysis, using variety of metrics and comparisons, are conducted and provided in Sections 4 and 5. Finally, further perspectives are discussed in conclusion.

4.1 Preliminaries

4.1.1 Notations

The notations used throughout this chapter are described in Table 4.1 as the following:

Table 4.1 – Notations used in this chapter

Notation	**Description**
$MA(C, d, S_x)$	Mobile Agent with code C, data d and state in x;
RP	Remote Platform;
ID_x	Identity of entity x;
AID_x	Anonymous identity of entity x;
p, q	Two large prime, such that: $q \mid p-1$;
G_1, G_2	A cyclic additive group and a cyclic multiplicative group with prime order q;
$e : G_1 \times G_1 \longrightarrow G_2$	An efficient admissible bilinear map;
F_p	Finite Field;
$E(F_p)$	An elliptic curve over F_p defined by the equation $y^2 = x^3 + ax + b$, where $a, b \in F_p$ and $4a^3 + 27b^2 \neq 0$;
G	The cyclic additive group of order q, which consists of points on $E(F_p)$ and an "infinite point";
P	a generator of G;
$H_i(.)$	Collision resistant cryptographic hash function
$\|$	Concatenation
$\oplus$	Exclusive-Or operation

4.1.2 Elliptic Curve Cryptography

Over a finite field F_p, an elliptic curve **[Kap08]** is defined by an equation of the form: $y^2 = x^3 + ax + b$, where a and b are arbitrary constants, satisfying the condition $4a^3 + 27b^2 \neq 0$. An elliptic curve defines O, a point at infinity, which serves as the identity element for some operations. O with all rational points of the curve form an abelian (commutative) group $E(F_p)$

under addition modulo p operation. The operations include the addition of two points and the double of a point.

Point Addition: given $P, Q \in G$, and l the line determined by P and Q in case $P \neq Q$ or by the tangent line to E/F_p in case $P = Q$. The sum $P + Q$ can be computed as the reflected point of R about the x-axis, such that R is the intersection between the curve E and the line l.

Point Multiplication: multiples of a point P can be viewed as repeated addition operations: $nP = \underbrace{P + P + \cdots + P}_{ntimes}$.

Elliptic curve discrete logarithm problem (ECDLP): given $P, Q \in E(F_p)$, find the integer k such that $Q = kP$.

Computational Diffie-Hellman problem (CDH): given a tuple $\{P, aP, bP\} \in G$ for some $a, b \in \mathbb{Z}_p^*$, the CDH problem in G is to compute the element abP.

In this work, the elliptic curve cryptography is associated with bilinear pairing, that is defined as follows:

Let G_1 and G_2 be a cyclic additive group generated by P of prime order q and a cyclic multiplicative group of the same order q, respectively. A map $e : G_1 \times G_1 \longrightarrow G_2$ is called a computable bilinear pairing [**Enge13**] if it is featured with the following properties:

1. **Bilinearity**: for every two random points $X, Y \in G_1$, and $a, b \in Z_q^*$, $e(aX, bY) = e(X, Y)^{ab}$.
2. **Non-degeneracy**: there exists $X, Y \in G_1$ such that $e(X, Y) \neq 1_{G_2}$.
3. **Computability**: for any two random points $X, Y \in G_1$, $e(X, Y)$ could be calculated by an efficient algorithm in polynomial time.

A bilinear map e can be implemented using the modified Weil or Tate pairing [**Enge13**] defined on elliptic curves.

4.1.3 Intrusion Detection System

Among the famous solutions to address security issues in an information system is to detect intruders performing malicious activities, through integrating intrusions detection system (IDS). This later is defined by the NIST [**Scarf07**] as a software or hardware device, potentially capable to identify an attack and notify appropriate personnel immediately, which help to stop possible threats or at least prevent them from succeeding.

According to [**Patel10**], an IDS is built with four functional layers as illustrated in Figure 4.1.

— **Infrastructure**: this means in general the technology layout, where the IDS is incorporated. This may concern either individual or collaborative structures, with different sorts of architectures: "centralized" where detection and alert elements are produced locally; "hierarchical" in which the system is divided into several small groups with similar features, such that lowest levels work for detection and highest ones for alert correlation, and "distributed" that ensures an autonomous system with distributed management control, where each participant possesses its own functional components to communicate with others.

— **Monitoring**: this is necessary related to the analysis of the collected data through monitoring the network traffic. This may contain particular segments and devices, dynamic behaviors stating which actions are performed on which resources, or events occurring on specific applications and which affect the overall performance. This analysis is

made according to the monitored environment, if it is either network-based, host-based, application-based or a combination that provides more high flexibility and efficiency.

— **Detection**: it is mostly occurred in real-time, while the system is monitored to depict abnormal activities and behaviors, and then provide adequate prevention solution. There are three main categories of detection methods. Misuse methods employ specifically traditional patterns such as: signatures to predict and detect intrusions. Anomaly methods inspect incidents on frequency and uncover abnormal patterns of behaviors, using variety of approaches such as:
  — "Statistical Patterns": they are based on observation of subjects activities and edition of behaviors profiles,
  — "Machine Learning": it is able to improve and optimize its performance over time and revise its strategy depending on the returned reactions,
  — "Data Mining": it makes the detection process more effective through deploying patterns, classifications, structures or events in data.

  Finally, Hybrid methods combine the both categories, such that misuse detects known attacks, while anomaly detects unknown attacks.

— **Response**: once an attack is detected, the system can follow two prevention strategies: "passive", in which the system can simply terminate its activity, and "active", in which the system generates correlated alarm clustered into incidents, blocks the activity of the malicious source and edits vulnerabilities reports with alert contexts.

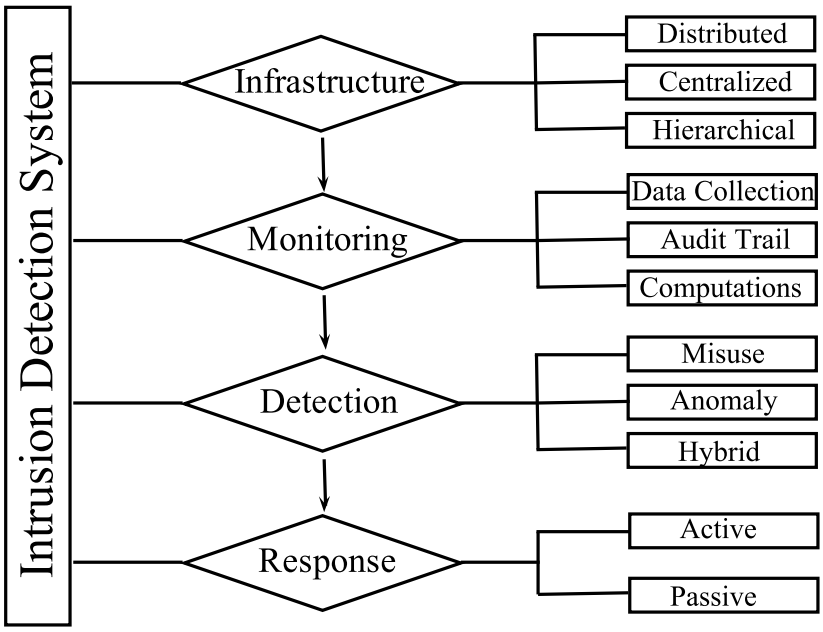

Figure 4.1 – IDS Layering and Functionalities

4.2 Anonymous Authentication using Elliptic Curve Cryptography

Authentication is one of the most important security mechanisms in any information system, that aims at ensuring safety of its entities and components. Among the variety of authentication mechanisms proposed in literature [**Lee07**], the *"Anonymous Authentication Schemes"* are recently proposed, and they are particularly provided with privacy preservation and anonymity in mobility networks. In this section, we propose a novel anonymous authentication scheme based on elliptic curve cryptography [**Kap08**], to ensure efficiency, reliability and resistance to a large range of attacks. Moreover, due to the limited resources and computing capacity of mobile agents, we make use of lightweight protocol where low cost functions (hash function, Or-exclusive, concatenation) are employed to reduce computation loads. At the end of the scheme, a session key is shared to encrypt and decrypt the exchanged messages between the agent and the platform. Indeed, there are three phases in the proposed scheme: the initialization phase, the registration phase and the authentication with key agreement phase. These phases are described in details as follows.

4.2.1 Initialization Phase

In this phase, the mobile agent MA generates all parameters needed by the system through the following steps:

1. MA generates randomly two large prime numbers p and q, then chooses an elliptic curve $E(F_p)$ over F_p.
2. MA generates an additive group G_1 with the order q, a multiplicative group G_2 with the same order q, a generator P of the group G_1 and a bilinear pairing $e : G_1 \times G_1 \longrightarrow G_2$.
3. MA chooses two secure hash functions
   — $H_1 : \{0,1\}^* \times G_1 \longrightarrow G_1$
   — $H_2 : \{0,1\}^* \times G_2 \longrightarrow \mathbb{Z}_q^*$
4. MA generates a random number as secret key $x_{MA} \in \mathbb{Z}_q^*$ and computes $Y_{MA} = x_{MA}.P$.
5. MA sends $\{E(F_p), G_1, G_2, H_1(.), H_2(.), Y_{MA}\}$ to the remote platform and keeps secret x_{MA}.

4.2.2 Registration Phase

In this phase, the remote platform RP securely registers for MA to be able to receive it and execute it safely. In order to prove the temporal correctness of the execution, we need to ensure time synchronization between MA and RPs. Thus, we make use of Network Time Protocol (NTP) [**Mills10**] to keep the clocks in sync. Figure 4.2 shows the steps of registration between MA and RP.

1. Once receiving the system parameters, RP generates a random number as secret key $x_{RP} \in \mathbb{Z}_q^*$ and computes $Y_{RP} = x_{RP}.P$. Then, RP submits his identity ID_{RP} along with Y_{RP} to MA over a secure channel (Secure Socket Layer (SSL)) .
2. MA checks the validity of ID_{RP}. If it is not valid, MA reports a conflict; otherwise, it produces a timestamp t_{MA} and two random ephemeral secrets $a, b \in \mathbb{Z}_q^*$. Then it computes $A_{MA} = a.P$, $B_{MA} = b.P$, $Q_A = A_{MA} \oplus Y_{RP}$ and $Q_2 = B_{MA} \oplus Y_{RP}$. Afterwards, using its secret key and the ephemeral keys, MA hides its real identity and that of RP

Remote Platform (RP)		**Mobile Agent (MA)**
Generates x_{RP} $Y_{RP} = x_{RP}.P$		
	$\xrightarrow{ID_{RP}, Y_{RP}}$	
		Generates a, b and t_{MA} $A_{MA} = a.P$; $B_{MA} = b.P$ $Q_1 = A_{MA} \oplus Y_{RP}$ $Q_2 = B_{MA} \oplus Y_{RP}$ $AID_{MA} = H_1(ID_{MA} \parallel t_{MA} \parallel x_{MA} \parallel A_{MA} \parallel B_{MA})$ $AID_{RP} = H_1(ID_{RP} \parallel Q_1 \parallel Q_2)$ $C_{RP} = a + b + AID_{RP}.P$
	$\xleftarrow{\{Req\_AID'_{RP}, A_{MA}, B_{MA}, AID_{MA}, C_{RP}\}}$	
Generates t_{RP} $Q_1 = A_{MA} \oplus Y_{RP}$ $Q_2 = B_{MA} \oplus Y_{RP}$ $AID'_{RP} = H_1(ID_{RP} \parallel Q_1 \parallel Q_2)$		
	$\xrightarrow{AID'_{RP}, t_{RP}}$	
		Checks t_{RP}, verifies: $AID_{RP} == AID'_{RP}$ $C_{RP}.P = A_{MA} + B_{MA} + AID'_{RP}.Y_{RP}$
	$\xleftarrow{VerificationStatus: On/Off}$	

Figure 4.2 – Registration Phase in the proposed approach

and computes the corresponding anonymous identities as follows:
- $AID_{MA} = H_1(ID_{MA} \parallel ts \parallel x_{MA} \parallel A_{MA} \parallel B_{MA})$, and
- $AID_{RP} = H_1(ID_{RP} \parallel Q_1 \parallel Q_2)$;
At last, MA computes $C_{RP} = a + b + AID_{RP}.x_{MA}$ and sends $\{Req\_AID'_{RP}, AID_{MA}, A_{MA}, B_{MA}, C_{RP}\}$ to RP.

3. Upon receiving $\{Req\_AID'_{RP}, AID_{MA}, A_{MA}, B_{MA}, C_{RP}\}$, RP produces a timestamp t_{RP} and computes $Q_1 = A_{MA} \oplus Y_{RP}$, $Q_2 = B_{MA} \oplus Y_{RP}$, and $AID'_{RP} = H_1(ID_{RP} \parallel Q_1 \parallel Q_2)$. Then RP sends AID'_{RP},t_{RP} to MA.
4. MA checks the validity and freshness of t_{RP}, then verifies whether $AID_{RP} == AID'_{RP}$ and $C_{RP}.P == A_{MA} + B_{MA} + AID'_{RP}.Y_{MA}$, where == refers to the comparison. If this verification does not match, the MA returns back an authentication reject message (Status: Off). Otherwise, MA advises RP to continue the authentication (Status: On).

4.2.3 Authentication and Key Agreement Phase

In this phase, the MA can achieve an anonymous authentication after registering RP. The procedure of the authentication and key agreement phase is shown as follows and illustrated in

Figure 4.3.

Remote Platform (RP)		**Mobile Agent (MA)**
Generates n and t'_{RP} $N_{RP} = n.P$; $M_{RP} = n.Y_{RP}$ $K_{RP} = H_1(N_{RP} \parallel M_{RP} \parallel t'_{RP})$ $U_{RP} = e(K_{RP}, M_{RP})$ $Z_{RP} = x_{RP} + (H_1(AID'_{RP} \parallel U_{RP} \parallel t'_{RP}))^{n.x_{RP}} modq$ $cipher = Enc_{K_{RP}}(AID'_{RP} \parallel Z_{RP})$	$\xrightarrow[cipher]{N_{RP}, t'_{RP}}$	
		Checks t'_{RP}, computes: $M'_{RP} = x_{MA}.N_{RP}$ $K_{MA} = H_1(N_{RP} \parallel M'_{RP} \parallel t'_{RP})$ $U'_{RP} = e(K_{MA}, M'_{RP})$ $clear = Dec_{K_{MA}}(cipher)$ $Z'_{RP} = clear + AID_{RP}$ Checks the equation: $Z'_{RP}.P = Y_{RP} + P.(H_1(AID_{RP} \parallel U'_{RP} \parallel t'_{RP})^{M'_{RP}.Y_{RP}/Y_{MA}} modq$ if it holds, generates d, computes: $D_{MA} = d.P$ $Pass = H_2(D_{MA} \parallel M'_{RP} \parallel Z'_{RP}.P \parallel t'_{RP})$ **Session Key:** $SK = H_2(Pass \parallel d.N_{RP})$
	$\xleftarrow[Pass]{D_{MA},}$	
Checks: $Pass = H_2(D_{MA} \parallel M_{RP} \parallel Z_{RP}.P \parallel t'_{RP})$ **Session Key:** $SK = H_2(Pass \parallel n.D_{MA})$		

Figure 4.3 – Authentication and Key Agreement Phase in the proposed approach

1. RP produces a second timestamp t'_{RP} and generates a nonce $n \in \mathbb{Z}_q^*$, then computes $N_{RP} = n.P$, $M_{RP} = n.Y_{MA}$, $K_{RP} = H_1(N_{RP}||M_{RP}||t'_{RP})$, $U_{RP} = e(M_{RP}, K_{RP})$, $Z_{RP} = x_{RP} + (H_1(AID'_{RP}||U_{RP}||t'_{RP})^{n.x_{RP}} modq$ and $cipher = Enc_{K_{RP}}(AID'_{RP}||Z_{RP})$. At last, $\{N_{RP}, cipher, t'_{RP}\}$ are sent to MA.
2. Upon receiving $\{N_{RP}, cipher, t'_{RP}\}$, MA checks the validity and freshness of t'_{RP}, then computes $M'_{RP} = x_{MA}.N_{RP}$, $K_{MA} = H_1(N_{RP}||M'_{RP}||t'_{RP})$ and $U'_{RP} = e(K_{MA}, M'_{RP})$. Using the key K_{MA}, MA decrypts the cipher to get $clear = Dec_{K_{MA}}(cipher)$, computes $Z'_{RP} = clear \oplus AID_{RP}$ and checks whether the equation (1) holds.

$$Z'_{RP}.P = Y_{RP} + P.(H_1(AID_{RP}||U'_{RP}||t'_{RP}))^{M'_{RP}.Y_{RP}.Y_{MA}^{-1}} \tag{4.1}$$

If the equation does not hold, MA stops the authentication process and sends a failed-authentication message to RP. Otherwise, MA authenticates successfully RP and generates $d \in \mathbb{Z}_q^*$, computes $D_{MA} = d.P$,$Pass = H_2(D_{MA}||$
$M'_{RP}||Z'_{RP}.P||t'_{RP})$ and the session key $SK = H_2(Pass||d.N_{RP})$. Then, MA sends $\{Pass, D_{MA}\}$ to RP.

3. RP computes $H_2(D_{MA}||M_{RP}||Z_{RP}.P||t'_{RP})$ and checks whether it is equal to $Pass$. If not, the session is stopped; Otherwise, the session key is calculated $SK = H_2(Pass||n.D_{MA})$.

4.3 Intrusion Detection based on Execution Tracing

Detection of anomalies in an information system becomes one of the most requested features to predict risks and prevent harms. In this section, we proceed to the description of our proposed IDS integrated during the mobility of the agent across the network. We keep using elliptic curve primitives to generate cryptographic traces of the agent execution.

According to the conception criteria of the detection intrusion systems, our proposed IDS is provided with a fully distributed infrastructure, where the monitored environment is a combined network-host based area. Besides, we adopt an anomaly detection technique based on a statistical analysis of the behaviors registered by the subjects and objects. This is mainly related to the nature of mobile agent systems, that are endowed with high scalability and less energy consumption, which contribute efficiently in ensuring fast and rigorous detection of known and unknown intrusions. Moreover, we choose to make the response of our IDS as significant as possible, through revoking the malicious entity and ending its activities once it does not satisfy the security requirements.

When a native platform needs services from other remote platforms, it creates a mobile agent, that will be in charge of moving across these platforms, executing the requested actions and returning back with the obtained results. The named agent may contain a stack of IP addresses of the platforms it has to visit, the data needed to be involved and the code of the activities to be performed. Figure 4.4 shows a preview of an XML file of a serialized mobile agent, that needs to move across three remote platforms and perform specified activities on each one.

In this agent file, many information are indicated such as: agent credentials (name, identity, size, location), its itinerary including the hosts to be visited and their characteristics, the assigned tasks with input data file and results log file as mandatory file resources, the encrypted code of the mobile agent, the mobility departure time and the java keystore file containing the session keys shared with authenticated hosts. During this round-trip, the mobile agent may be victim of a variety of intrusion attacks, aiming at harming its code, data, status and path. Thus, it is strongly requested, that the agent records all behaviors and actions during its execution. Accordingly, when receiving the returning agent, the native platform will be able to detect maliciousness and anomalies occurred.

Tracing the activities of a mobile agent across different platforms was introduced by Vigna in [**Vigna98**], which was of a great inspiration for our contribution. According to Vigna's work, a trace is generated through a post-mortem analysis of data and events, during the execution of the agent code. The later is considered as a segment comprising a sequence of statements, which can be classified as white and black. White statement modifies the agent's execution using agent's internal variables only. For example: $a = b + c$ is a white statement, such that a, b and c are local variables of agent's code. Black statement makes use of information received from

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:Agents xmlns:ns2="http://NPMachine">
  <ns2:Agent>
     <queueSize>16</queueSize>
     <ns2:ID> MA_NP_4 </ns2:ID>
     <ns2:Name> MobAg_1 </ns2:Name>
     <ns2:Location> 192.168.252.46 </ns2:Location>
     <ns2:Itinerary>
        <group type="WINDOWS">
           <Hosts>
              <First name="RP1.domain.com" IP= "192.168.252.111" port="80"/>
              <Second name="RP2.domain.com" IP= "192.168.252.132" port="80"/>
              <Third name="RP3.domain.com" IP= "192.168.252.208" port="80"/>
          </Hosts>
       </group>
    </ns2:Itinerary>
    <ns2:Job>
       <task class="RoundTrip.CyclicActivities">
          <workspace ref="default"/>
          <resource>input.dat</resource>
          <resource>result.log</resource>
       </task>
    </Job>
    <ns2:code format="base64">c314cd0ca81dcd1bdd13e80d5948a2fdd757a79ca
               8741237e34e7859e4fd2cf2940191f2a798b79aecb8caa081e283354379
               dd68f802c01fb0d16de128da8252271cb234687d67ddd9cb7a4280203a
               b9040057af30a764b1f6c1c5756a83a5fb77ce797b15e0dd3499132c9c1
               ad0bd50f1913520fe83b3ec4c2d4cac183969a528b53abfaaebfe0368725
               2ddf9636b34c2879d204c301c89252d3690cf09263070.....</ns2:code>
    <ns2:date> 2015-08-13 18:05:11.234 GMT</ns2:date>
    <ns2:KeyStore>
       <Secure OpenSSL="true" >
         <resource>NP_KeyPair.jks</resource>
         <resource>SessionKeys.jks</resource>
       </Secure>
    </ns2:KeyStore>
  </ns2:Agent>
</ns2:Agents>
```

Figure 4.4 – XML Code of serialized mobile agent with three destinations in its itinerary

external environments to alter the state of the program, such as a function $read(x)$ that assigns to x a value obtained from the terminal. Actually, the produced traces enable the verification of the agent execution, when malicious entities aim at tampering the agent, through claiming and supposing fake scenarios or operations.

The main idea of our approach is illustrated in Figure 4.5. Thus, during the round-trip of the mobile agent across n platforms (RPs) to execute n activities (Act), these later are recorded using a tracing function, that involves many parameters such as: the session key obtained at the authentication process, the identities of the sender, the receiver and the intended next recipient, timestamp and many others. In practice, the trace of the agent's execution on a remote platform

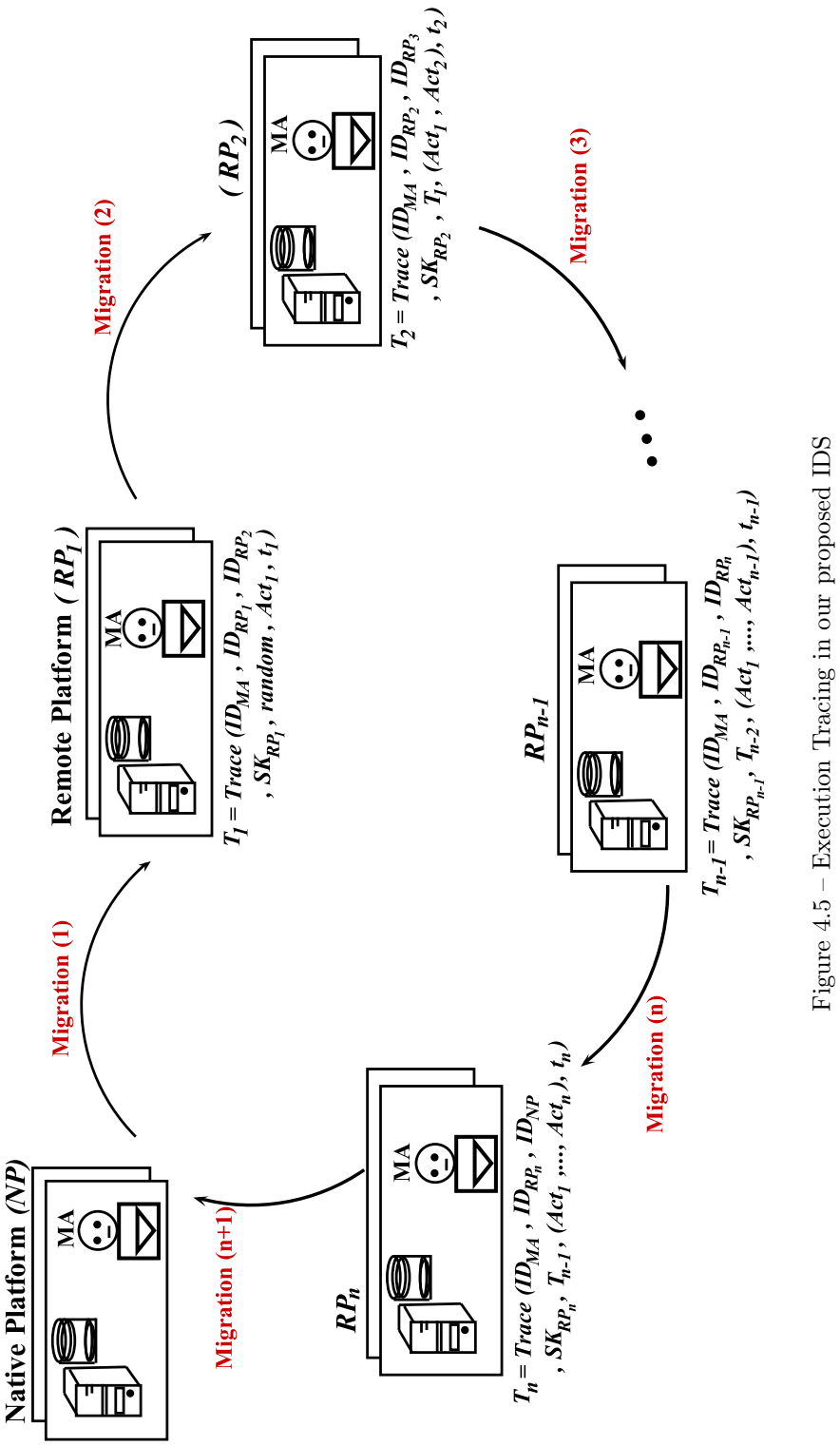

Figure 4.5 – Execution Tracing in our proposed IDS

is represented by the pair: $T =< U, M >$. U is the unique session identifier of the trace obtained through combining the session key with a unique identifier, which is randomly generated using the Java package java.util.UUID. While M is an encapsulated message containing many fields with signatures and encryption, and which are associated with the black statement of the executed code (Activities). This message is formatted as follow:

— The first and second fields (ID_{NP}, ID_{RP_i}) indicate respectively the identities of the agent owner and sender.

— The third field (ts) indicates a timestamp referring to the arrival time of the mobile agent to the platform RP_{i+1}.

— The fourth field contains the AES-256 encryption [**Rober12**] of a data set using the session key SK_i obtained through the authentication. This set includes two values:

1. The elliptic curve digital signature (ECDSA) [**Kap08**] of the following platform identity $(ID_{RP_{i+1}})$ and the timestamp (ts), using the ECC secret key of the current platform (x_{RP_i}) calculated during the authentication.
2. The ECC encryption using the public key of the next platform $(Y_{RP_{i+1}})$ of a data set, that includes the unique identifier of the trace (u), the agent code, the black statement executed in the code (BS), the results obtained (RT) and the set of traces generated on the visited platforms $(random, T_1, ..., T_{i-1})$.

— The fifth field includes the ECDSA signature using the agent's owner private key (x_{NP}) of the hash values of $(BS$ with $RT)$ and (T_{i-1}).

On RP_1 :=> to RP_2:
$\rightarrow M_1 := (ID_{NP}, ID_{RP_1}, ts, Enc_{AES_{SK_1}}(Sign_{ECDSA_{x_{RP_1}}}(AID_{RP_2}, ts),$
$Enc_{ECC_{Y_{RP_2}}}(u, Code, random)), Sign_{ECDSA_{x_{NP}}}(hash(random)).$

On RP_i :=> to RP_{i+1}:
$\rightarrow M_i := (ID_{NP}, ID_{RP_i}, ts, Enc_{AES_{SK_i}}(Sign_{ECDSA_{x_{RP_i}}}(AID_{RP_{i+1}}, ts),$
$Enc_{ECC_{Y_{RP_{i+1}}}}(u, Code, BS, RT, (random, T_1, ..., T_{i-1}),$
$Sign_{ECDSA_{x_{NP}}}(hash(BS, RT), hash(code), hash(T_{i-1})).$

After being executed on the current hosting platform "RP_i" and before moving to its following destination "RP_{i+1}", the mobile agent must authenticate this latter, and then produce the trace of its current execution $T_i =< U_i, M_i >$. When the remote platform "RP_{i+1}" receives the mobile agent with its trace, it begins by verifying the message M before proceeding to the execution of the agent on its environment. First of all, the identities of the agent's owner and sender are verified, and the timestamp (ts) is checked for freshness. Then, using the session key (SK_{i+1}) obtained at the authentication process, the fourth field of M is decrypted using the symmetric-key algorithm AES-256. Thus, to be sure that the trace is valid and exactly intended to it, "RP_{i+1}" decrypts the signature $Sign_{ECDSA_{x_{RP_i}}}(AID_{RP_{i+1}}, ts)$ using the public key of the sender. The anonymous identity owned by "RP_{i+1}" is compared with that provided in the signature, without forgetting to check ts and its compatibility with the common reference clock.

Once the comparison is valid, "RP_{i+1}" decrypts the ECC encryption using its private key, and gets in clear: the unique identifier of the trace, the agent code, the black statement executed on "RP_i", the obtained results and the set of traces produced from the launching of agent till its execution on "RP_{i-1}" (for the trace on the first remote platform and in the absence of a previous trace, this latter is replaced by a random, generated on "NP"). Accordingly, in order

to check the integrity of the mobile agent, we make use of the one-way hash function SHA-3 [**Jaffar13**], to separately hash the agent code, the executed statements along with the obtained results, as well as the previous trace on "RP_{i-1}". These hash values are compared to those contained in the signature included in the fifth field of M, and which is decrypted using the public key of the agent's owner (Y_{NP}).

As we adopt a chaining mechanism to link the traces, it becomes sufficient for "RP_{i+1}" to verify the last trace T_i coming from "RP_i". Finally, once passing all these verification, the platform can execute the mobile agent according to the required specifications and then, produce the relevant trace after authenticating the next destination. Otherwise, the agent edits a failure report, ends its activity on this platform and moves to the following one. When all the hosts in the agent itinerary are visited, the mobile agent returns back to its native platform "NP", with the obtained results and the tracing of the executions performed. In case that "NP" suspects an execution or an alteration of the agent on a particular platform, it consults the relevant cryptographic trace and inspects it deeply. in case it still has doubts, "NP" sends a mobile agent in order to re-execute the same specified tasks and produce a new cryptographic trace to be compared with the old one. If the comparison does not match, then an intrusion is detected.

4.4 Security Analysis

Anonymity

In our scheme, the real identity of the mobile agent is contained in an anonymous identity $AID_{MA} = H_1(ID_{MA} \parallel ts \parallel x_{MA} \parallel A_{MA} \parallel B_{MA})$, such that: $A_{MA} = a.P$ and $B_{MA} = b.P$ with a and b two nonces for randomization and freshness. In addition to the fact that ID_{MA} is protected in a hash function, an adversary that intercepts the messages between MA and RP will not be able to calculate the real identity of MA, as he will face the CDH problem and because the two nonces are unknown for him. Therefore, our proposed scheme maintains the agent anonymity.

Untraceability

untraceability is called to be provided when an adversary can not identify the real identity of the mobile agent MA or make a link between any two of its sessions. An adversary can collect by eavesdropping all the anonymous identities of MA (AID_{MA}) from the exchanged messages in every session. If the agent preserves the same value of AID_{MA}, the adversary may guess that this value frequently requested is containing the identity of the agent, and can employ it without recognizing its real value. In other words, the adversary traces that a same specific value is continuously requested at the same time and position in the authentication process. In our scheme, due to the unique timestamp and the random ephemeral secrets, all the produced anonymous identities of MA are different and the adversary cannot trace who communicates with MA by monitoring the channel. Therefore, our proposed scheme provides the attribute of untraceability to users.

Perfect Forward Secrecy

Perfect forward secrecy is called to be provided when the previous session keys cannot be disclosed, even if the secret keys of the mobile agent (MA) and the remote platforms (RP) are compromised. In our proposed scheme, the session key is $SK = H_2(Pass \parallel n.d.P)$ with n and

d two ephemeral secrets chosen by MA and RP, which guarantee key freshness in each session. Despite the fact that an adversary could hold the secret keys of both MA and RP, he will not be able to get the session key as he still needs at least to compute $n.d.P$ from $N_{RP} = n.P$ and $D_{MA} = d.P$, which is impossible due to the intractability of the Computational Diffie−Hellman (CDH) problem. Thus, the perfect forward secrecy is proven.

Known Key Security

Known Key Security is called to be provided when a produced session key is compromised and it could not influence the other session keys. In our proposed scheme, we can clearly see that an intruder cannot determine any other session key given a corrupted one, since these session keys are independently computed using the random ephemeral secrets n and d, which are separately generated and fairly contributed for every session. Therefore, our proposed scheme could provide the known-key security.

Man-In-The-Middle Attack

In our scheme, the mobile agent MA authenticates the remote platform RP through verifying if the equation(1) holds. Without the private key of RP: x_{RP} and the ephemeral secrets n, any adversary cannot generate N_{RP} neither K_{RP} to compute Z_{RP} and the cipher. In the same way, RP authenticates MA through verifying if the equation: $Pass = H_2(D_{MA} \parallel M_{RP} \parallel Z_{RP}.P \parallel t'_{RP})$ holds. Without the secret key of MA: x_{MA}, an adversary cannot compute M'_{RP} from N_{RP}, which means he cannot compute the accurate value of $Pass$. Accordingly, our scheme provides a mutual authentication between the communicating parties, which makes it well-resistant to the Man-In-The-Middle attack.

Replay Attack

In a replay attack, an adversary may gather the exchanged messages over a public network, and attempt to replay them for the other entities implicated in the authentication process. Our scheme can resist this attack due to the use of timestamp and ephemeral secrets. Even if an adversary intercepts messages from MA and RP, he cannot replayed to this messages to be authenticated with MA or RP since these latter contribute fairly in generating random nonces for each authentication session. In addition, the freshness of the timestamps produced for every session can be easily verified.

4.5 Performance Analysis

In this section, an analysis of our approach performance is provided according to various metrics: time, detection rate, scalability and effectiveness. For that purpose, JADE framework is used for the implementation, that considers a set of heterogeneous machines with different operating systems (Windows, Ubuntu, MacOs), such that only one is viewed as "Native Machine" and the others as "Remote Machines". The technical characteristics of these machines are: Intel Core i7 processor at 2.7 GHz with 4 Go of RAM, connected through 100Mbps switched Ethernet network with WampServer. Concerning the characteristics of the mobile agent as an entity with limited resources, it has a storage memory of 512 Mbits and GenuineIntel 800 MHz processor with HTTP-based-MTP (Message Transport System).

4.5.1 Authentication Performance

In this part, we evaluate the computational cost of our anonymous authentication scheme and we compare its performance with other well-known schemes, like Liu et al.'s schemes in its two versions: preliminary and security-enhanced [**Liu14**], Xiong's certificateless remote anonymous authentication scheme [**Xiong14**], as well as Zaho's scheme for wireless networks [**Zhao14**]. To be compatible with those named schemes and to achieve the comparable level of security to 1024-bits RSA, we employed the super-singular elliptic curve $E/F_p : y^2 = x^3 + x$ providing groups with a Tate pairing $e : G_1 \times G_1 \longrightarrow G_2$. The embedding degree of this curve is 2, q is a 160-bit prime and p is a 512-bit prime generated using the cryptographic pseudo random generator ISAAC+ [**Auma06**]. For convenience, we define some notations as follows:

— T_e: the running time for performing a module exponentiation operation;
— T_h: the running time for performing a one-way hash function;
— T_p: the running time for performing a bilinear pairing operation;
— T_m: the running time for performing an elliptic curve point multiplication;
— T_a: the running time for performing a modular addition/subtraction operation;
— T_s: the running time for performing a symmetric encryption/decryption operation;

Table 4.2 – Computational cost (in ms) of the operations performed on Remote Platform (RP) and Mobile Agent (MA)

Operation	RP	MA
T_e	8.75	19.43
T_h	0.07	0.26
T_p	17.81	92.20
T_m	4.63	11.38
T_s	10.32	14.84

Table 4.2 shows the computational cost of the different operations involved in the authentication process. Indeed, the running time of these operations either on the remote platform RP or the mobile agent MA is derived through repeated simulation experiments. Eventually, the total cost of the authentication phase in the proposed scheme is separately formulated as follows:

— *On RP*: $3T_p + 3T_h + 2T_m + T_a + T_s + T_e$
— *On MA*: $3T_p + 4T_h + 2T_m + T_a + T_s + T_e$

From comparative perspective, Table 4.3 exposes the computational cost of the three mentioned schemes compared to ours, that shows a better computational efficiency.

Furthermore, Figure 4.6 illustrates the comparison of the communication overhead between the schemes of Liu et al [**Liu14**], Xiong [**Xiong14**], Zhao [**Zhao14**] and ours. This comparison relies on the time consumed by the mobile agent (as a service requester) through the processing of authentication operations over an increasing number of visited remote platforms (as service providers). Besides, this overhead is basically evaluated depending on the size and duration of the request/response messages, that are exchanged during the authentication processes in one round-trip of the mobile agent. The results indicate that our protocol reduces at least 80%, 67% and 23% from the overall running time of Liu et al, Zhao and Xiong schemes, respectively.

Table 4.3 – Computational cost of our scheme compared to the schemes of Liu et al, Xiong and Zhao

	RP (service provider)	MA (service requester)
Liu et al [**Liu14**]	$T_p + 3T_h + 2T_m + 2T_e$	$4T_p + 3T_h + T_a + T_e$
Xiong [**Xiong14**]	$5T_p + 4T_h + 2T_m + T_a$	$4T_p + 5T_h + 3T_m + T_a$
Zhao [**Zhao14**]	$6T_p + 5T_h + T_s$	$3T_p + 4T_h + T_m + T_a + T_s$
Our Scheme	$3T_p + 3T_h + 2T_m + T_a + T_s + T_e$	$3T_p + 4T_h + 2T_m + T_a + T_s + T_e$

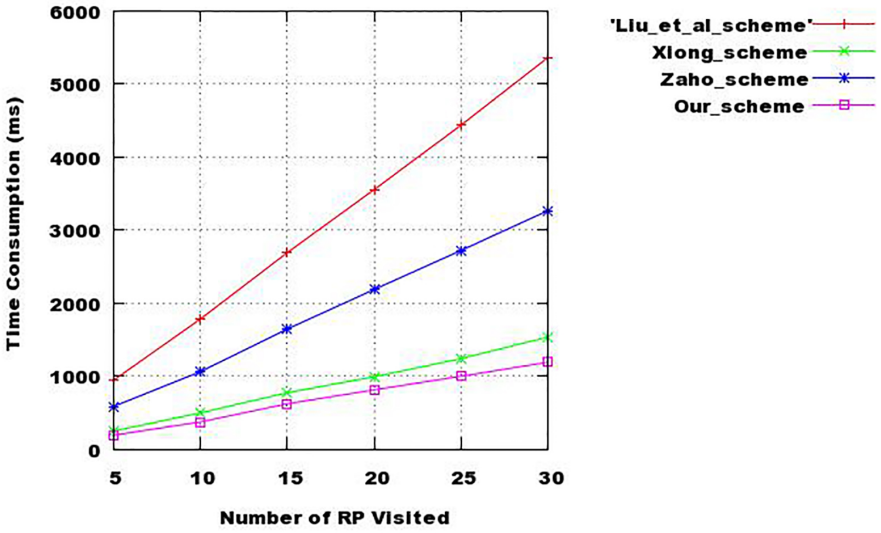

Figure 4.6 – Time consumption of the authentication process when a MA moving across an increasing number of RPs

An evaluation of the energy consumption of our scheme compared to the other three ones is provided in Figure 4.7. Assuming, that a MA runs on a 1.6 MHz processor, its energy consumed can be calculated as $e_A = t_A \times C$, where t_A is the total computation time for authentication, and C is the CPU maximum power. The comparison, in terms of energy usage as shown in Figure 4.7, indicates that our scheme achieves the best performance, such that it reduces approximately 70%, 48% and 62% of the overall energy consumed by the schemes of Liu et al, Xiong and Zhao, respectively.

4.5.2 Detection Performance

The effectiveness of an IDS is assessed on how the detection method is capable to make correct attack detection. This section describes the experimental results and performance evaluation of the proposed IDS, according to many evaluation metrics: accuracy, overhead and consumption. For that purpose, we consider the confusion matrix in Table 4.4, where four reactions are considered given the real event and the IDS prediction:

— *True Negative (TN)*: correct IDS operation, where a real event is successfully defined as

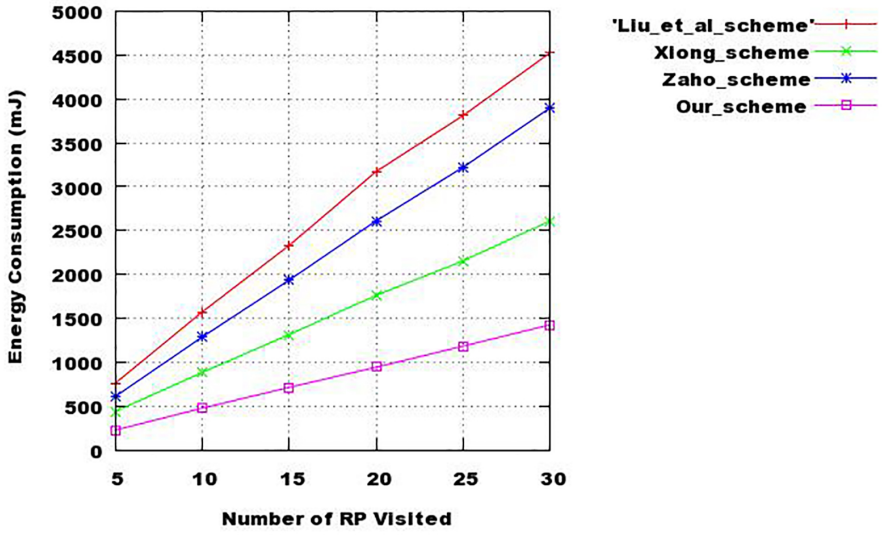

Figure 4.7 – Energy consumption of the authentication process when a MA moving across an increasing number of RPs

normal.

— *True Positive (TP)*: correct IDS operation, where a real event is successfully labeled as an attack.
— *False Negatives (FN)*: incorrect IDS operation, where a real attack is predicted as normal event.
— *False Positives (FP)*: incorrect IDS operation, where a normal event is predicted as an attack.

Table 4.4 – IDS Reaction in normal and intrusion situations

Predicted Situation	**Real Situation**	
	Normal	*Intrusion*
Normal	True Negative (TN)	False Positive (FP)
Intrusion	False Negative (FN)	True Positive (TP)

According to **[Blasco10]**, many measurements can be performed to quantify the effectiveness of an IDS, such as:

False Alarm rate (FAR):

$$\frac{FP}{TN+FP} = \frac{\text{number of false alarms}}{\text{number of alarms}} \tag{4.2}$$

Detection rate (DR) or Sensitivity or Recall (R):

$$\frac{TP}{TP+FN} = \frac{\text{number of detected attacks}}{\text{number of attacks}} \tag{4.3}$$

Accuracy (A):

$$\frac{TN+TP}{TN+FP+FN+TP} \tag{4.4}$$

Precision (P):

$$\frac{TP}{TP + FP} \tag{4.5}$$

Table 4.5 – Simulated Attacks using Metasploit

Name of Attack	**Description**
Nmap TCP Scan	performs a scan of a remote machine to determine the availableports that can be exploited to gain shell access of the server hosting the mobile agent.
Persisent Meterpreter Backdoor	given a listener payload on an open port of a remote machine, the attacker obtains almost complete control over the machine hosting the mobile agent.
DoS	using a spoofed address, the attacker applies a TCP flooding to burden the machine and make it unavailable for the agent.
Finger Service User	gives an enumerated wordlist of present users using smtp-enum module and allows an attacker to disrupt a network using the redirection capability in the finger daemon.

All experiments are carried out on heterogeneous machines equipped with JADE platform and the java library JPCAP (Network Packet Capture Facility) [**JPCap**] for capturing and sending packets. Real attacks are simulated and injected in the running environments using MetaSploit tool [**Mayn11**] dedicated for the penetration tests and the creation of secure solutions. In this context, we are concerned about evaluating the scalability criteria, such as response time and bandwidth consumption, as well as the effectiveness criteria in terms of detection rate and false alarm rate. The obtained results are compared to the centralized IDS SNORT tool [**Snort**], and to the system of Brahmi et al [**Brah11**], in which a Distributed Intrusion Detection using Mobile Agents and Snort (DIDMAS) is proposed.

In order to be compatible in the comparison with DIDMAS and SNORT, Table 4.5 defines the attacks adopted in our tests.

Scalability Evaluation

Among the very common and important metrics for evaluating a scalable system, there is the "response time" that defines the total time spent to depict an attack since its launching, as well as the "energy consumption" in terms of transmission capacity of network (bandwidth).

Figure 4.8 shows the response time achieved by our IDS, compared to DIDMAS and SNORT, in order to discover the four types of attacks indicated before. Thereby, we remark that our IDS expresses a fast response time behavior. In addition, as depicted in Figure 4.9, the bandwidth consumption of our IDS shows a low rate regarding DIDMAS and SNORT. This is mainly due to the mobility feature of the agent, that begins the analysis of data on the environments where attacks are located. The obtained results attest clearly the performance of our IDS, which will neither be compromised by the increase of attacks in type and number, nor by the increase of the remote hosts being visited. This is illustrated in Figure 4.10, that gives the time overhead added by our IDS while the number of visited hosts is rising.

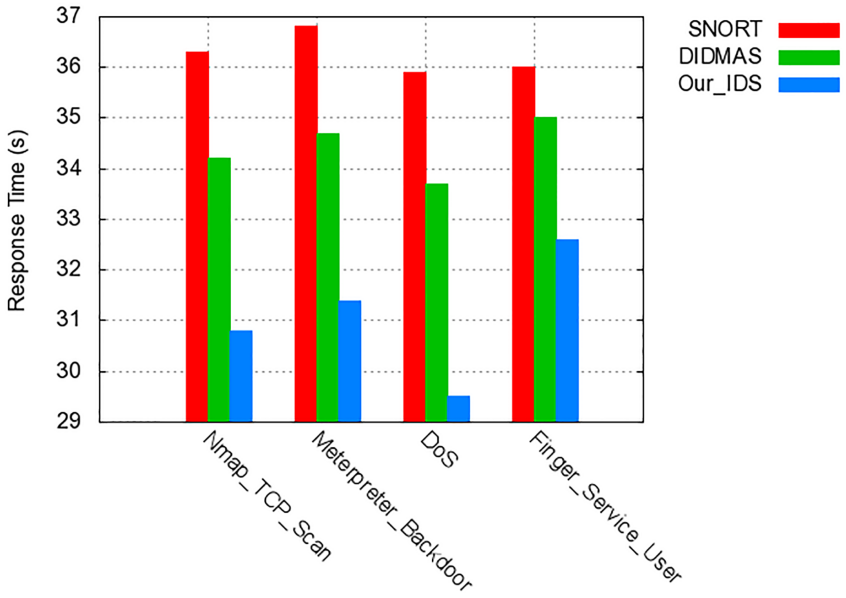

Figure 4.8 – Time Response of our IDS compared to DIDMAS and SNORT

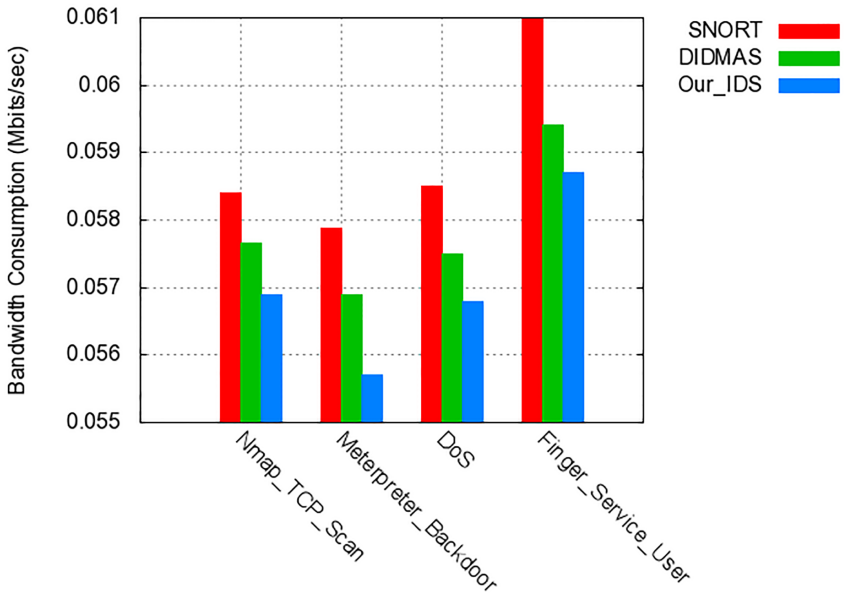

Figure 4.9 – Bandwidth Consumption of our IDS compared to DIDMAS and SNORT

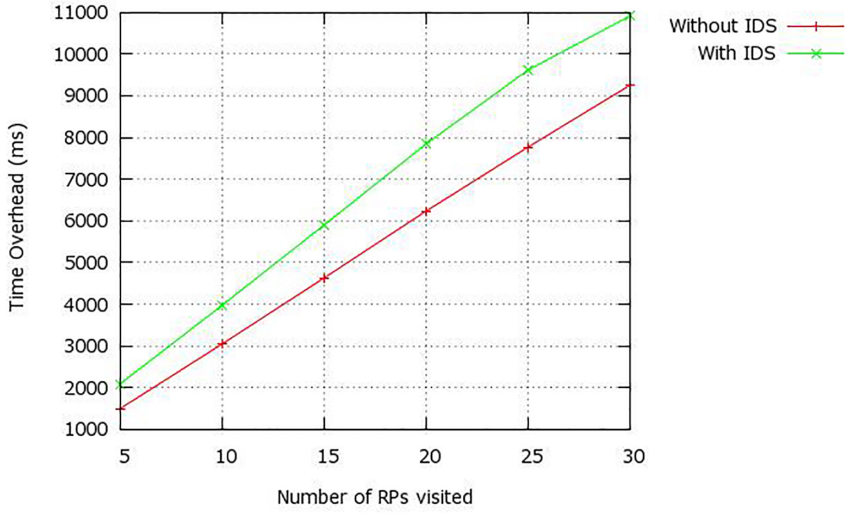

Figure 4.10 – Time overhead of our IDS regarding the increase of the visited hosts

Effectiveness Evaluation

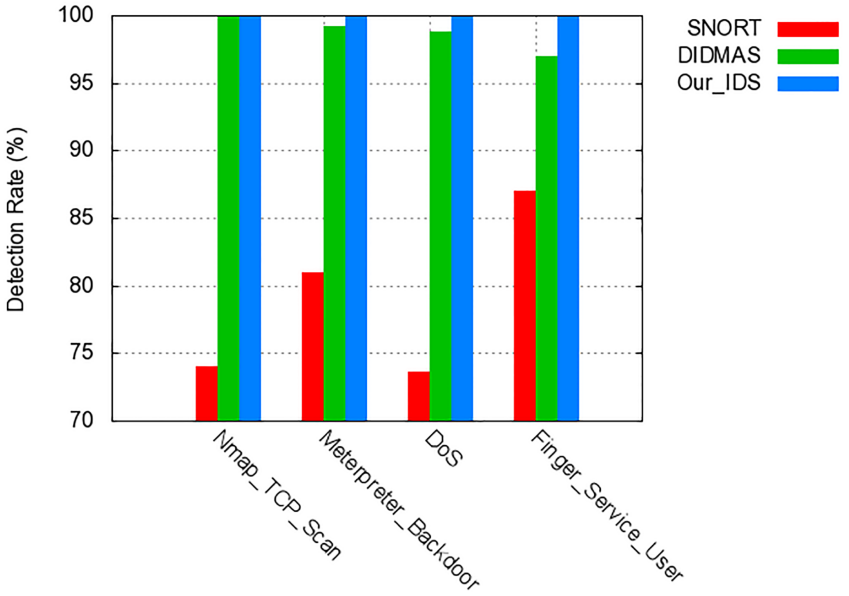

Figure 4.11 – Detection rate of our IDS vs DIDMAS and SNORT

The commonly used metrics to test the effectiveness of an IDS are: detection rate and false alarm rate. The first one, indicates the number of correctly detected attacks, while the second

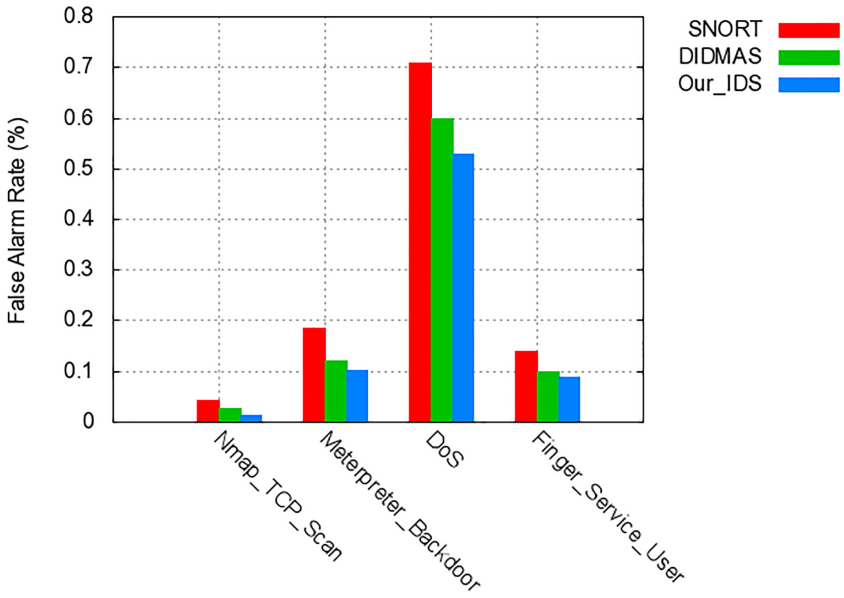

Figure 4.12 – False Alarm rate of our IDS vs DIDMAS and SNORT

indicates the normal scenarios incorrectly considered as intrusions. It is worth to notice, that a good IDS must have a high detection rate, whereas the false alarm rate is expected to be as low as possible.

This is well experienced in Figure 4.11, where our IDS shows notable large detection rates in comparison with DIDMAS and SNORT. These rates are improved in average of about 2% and 21% from DIDMAS and SNORT, respectively. Furthermore, Figure 4.12 indicates, that the false alarm rates resulting by our IDS are significantly lower of about 22% and 43% than DIDMAS and SNORT, respectively.

Thus, regarding the challenges that an IDS faces to prove its effectiveness, our IDS fits the existing requirements through providing a low false alarm rate and maintaining a high detection rate.

4.6 Conclusion

In this chapter, we have proposed a robust and distributed approach to address the security issues faced by mobile agents, during their migration. Making use of the cryptographic elliptic curve functions based on bilinear pairing, we have provided our agent with an efficient anonymous authentication and proactive execution tracing techniques. The carried out experimental results prove the scalability and effectiveness of the introduced approach, since it reduces the computational costs, in terms of response time and bandwidth consumption, and it exposes a high detection rate versus a low false alarm rate. As perspectives, we will be concerned in extending the IDS into an intrusion detection and prevention system (IDPS), as well as in deploying access control techniques for trustworthy platforms.

Chapter 5

Access Control and Cryptography for Agent Platforms

Contents

In this chapter, a novel contribution is introduced. It allows the hosting platform to grant access only to the authenticated mobile agents, and ensures secure communication between the both via the generation of a session key, that is used to encrypt and decrypt the exchanges. Besides, it monitors the accesses to the resources of the platform in a flexible and interoperable manner, which guarantees the confidentiality and lessens damages in case that a malicious agent is received.

This chapter is organized as follows. Section 2 provides an overview of the mechanisms utilized to conceive the proposed approch: 1) the access control principle and its famous models, and 2) the secret sharing principle based on threshold. In Section 3 the overall architecture of the approach is presented. In order to protect the hosting platform from agent attacks, Section 4 decribes the adopted authentication process, which is strengthened using a fixed and an enhanced version of Diffie-Hellman key exchange, While Section 5 presents our security policy to manage the access to the platform resources, through using an extended Discretionary Access Control model (DAC) along with Shamir-threshold sharing scheme. A security analysis based on variety of attacks is provided in Section 6. Moreover, in Section 7, the security, feasibility and effectiveness of the proposed contribution were evaluated according to three factors: time performance, authorization performance and the resistance to some well known attacks. Finally, a results interpretation and perspectives are provided in the conclusion.

5.1 Preliminaries

5.1.1 Access Control Policies

An access control policy is defined by Samarti [**Sam01**] as a set of high level directives that specify which entity (a subject) has the permission to exercise what operation (an action) on which data (an object). Thus, the three essential concepts on which an access control policy is based are:

— *Subject*: it is an active entity able to get access to the system assets. It can be a user, an application, a network server...

— *Object*: it is a passive entity representing the assets to be protected. It can be a file, a program, a device, a connection...

— *Action*: it is a particular operating mode processed by a subject on an object. For instance, the actions to be considered by a user on a file are: write, read and execute.

In view of its efficiency in preserving the security of an information system, especially its confidentiality and integrity, an access control policy must satisfy the following specifications and principles [**Sam01**]:

— **Least Privileges**: it consists in providing a subject with the access to the smallest number of objects needed to accomplish a task, even if it holds more permissions.

— **Verification of each access**: all the accesses granted to a user over an object must be verified continuously, even after that this user is authorized.

— **Tracking**: in addition to the implementation of the access control policy, its administration requires a parallel monitoring to know whether the access control policy is properly working or not.

— **Granularity**: it represents the smallest unit to which access is controlled. This can be a file, a program, a block of data, hardware devices, etc. In reality, everything depends on what to control with respect to users and their access.

— **Access Log Files**: the accesses authorized in a system are recorded in "Access/Audit Log" files, which allows to enhance the system features, identify the improper use of an object and the bugs sources as well as to detect penetrations.

According to a swot analysis on access control models conducted by M. Ennahbaoui et al [**Ennah14**], there are two major categories of access control models. The first category concerns the classical and basic models: Discretionary Access Control (DAC) and Mandatory Access Control (MAC). In the second category, enhanced and extended models are derived from the first category, such as: Role-Based Access Control (RBAC), Organization-Based Access Control (OrBAC) and many others. In this chapter, we make use of DAC model, hence it will be interesting to briefly define this latter.

DAC [**Ennah14**] is a flexible model that allows a subject to assign permissions to other subjects, and where the agreement or revocation of privileges is regulated by administrative policy. An example of this model usage is the management of access files in Unix operating system. There exist two famous discretionary models: Lampson model and Harrison Ruzzo Ullman (HRU) model.

In the Lampson model, the access rights are specified in a access control matrix. It is represented by the triplet (S, O, M_{SO}), where S is the set of subjects, O is the set of objects and M_{SO} is the access control matrix, that associates to each couple $(subject, object)$ a set of access rights, according to the nature of the object. Moreover, the matrix can be continuously updated through the creation or destruction of subjects and objects, as well as by the addition or removal of access rights. In practice, to overcome the storage problems, there are two applications of

this matrix: *Access Control List* where the actions are stored by column, and *Capabilities List* where the actions are stored by row.

The HRU model is an extension of the Lampson model. It makes use of the classical access control matrix with the advantage of specifying a set of a primitive operations, called "Commands", to assign the access rights, as well as to create and delete subjects and objects. In this model, when a subject S is the owner of an object O, then S can assign its access rights of O to other subjects. The commands are constructed from primitive operations such as: *enter*, *delete*, *create object*, *delete object*, etc.

5.1.2 Threshold Sharing Scheme

Let us consider an example of a Bank vault which must be opened every day. In the Bank, there is a category of employees who are trusted enough to participate in the opening of the vault, but not trusted if each one owns the complete combination to the vault. Hence, it would be reliable to design a system where for example any four of these employees together can open the vault, but no individual alone can do so. The above problem can be solved by means of a secret sharing scheme.

Secret sharing schemes were independently introduced by Blakley [**Blakl79**] and Shamir [**Sham79**] in 1979. In this chapter, we consider a special type of secret sharing scheme, called a threshold scheme. An informal definition of a threshold scheme is as follows:

Let k and n be positive integers, such that $k \leq n$. A (k, n)-threshold scheme is a method of sharing a secret K among a set of n participants in such a way that any k participants can compute the value of the secret , but no group of $k-1$ or fewer can do so.

According to this definition, when a dealer wants to share a secret among the participants, it gives each one some partial information calles "*share*". The most popular construction of these $(k, n)-$schemes is *Shamir Threshold Scheme*, which is formulated as follows:

Notations:

— let $p \leq n-1$: a prime number
— $P = \{P_1, P_2, ..., P_n\}$: the set of the participants
— $K \in \mathbb{Z}_p$: the possible secret
— $S \in \mathbb{Z}_p$: the set of possible shares
— D: a dealer

Initialization:
D chooses n public, distinct and non-zero elements from $\mathbb{Z}_p$, denoted x_i, such that: $1 \leq i \leq n$.

Distribution:

1. D wants to share the secret $K \in \mathbb{Z}_p$. D chooses randomly $k-1$ elements from $\mathbb{Z}_p$, denoted $a_1, a_2, ..., a_{k-1}$, such that: $a_0 = K$.
2. D computes $y_i = a(x_i)$, for $1 \leq i \leq n$, where: $a(x) = \sum_{j=0}^{k-1} a_j x^j \ mod\ p$
3. D gives the participant P_i the share y_i

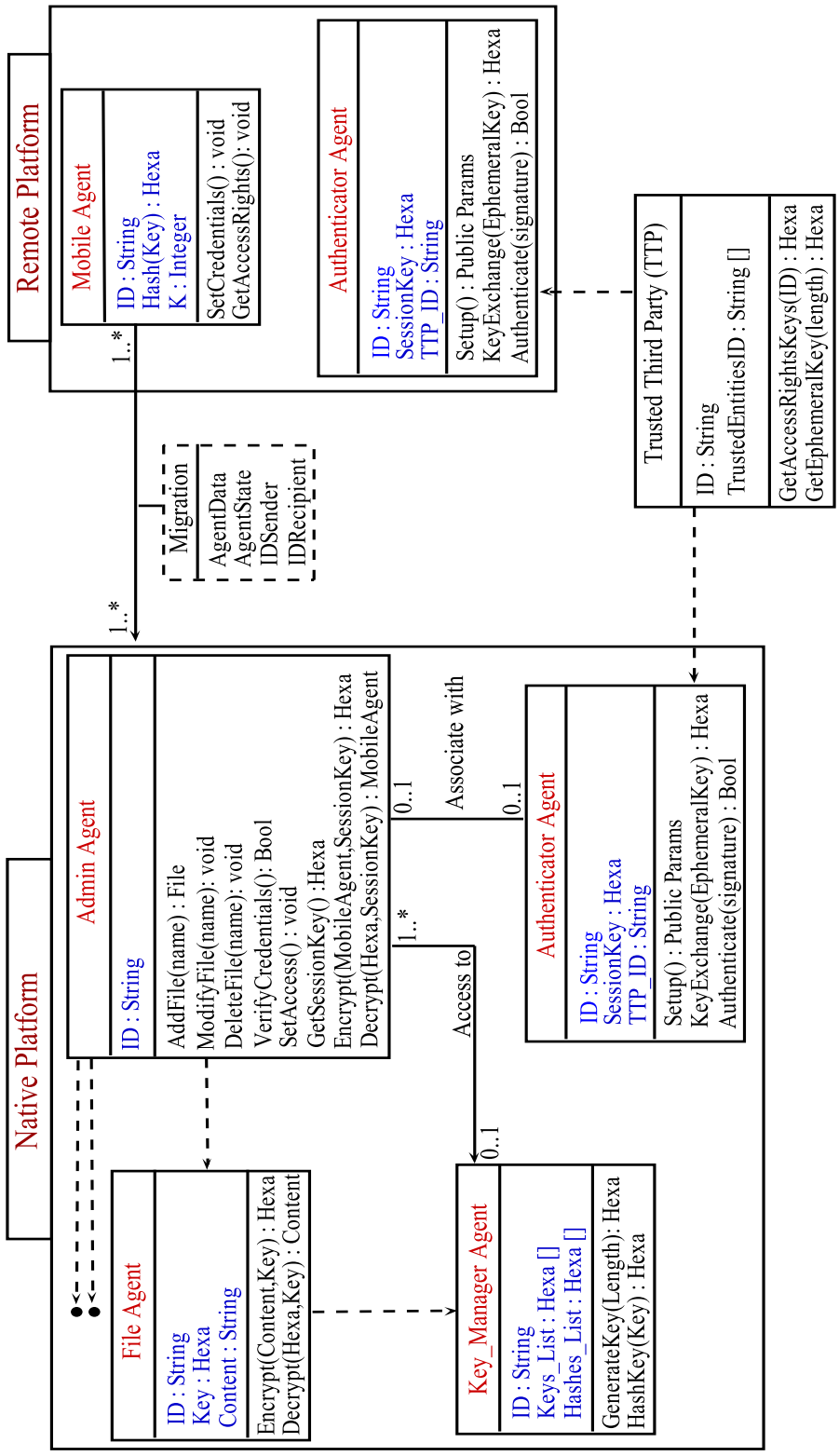

Figure 5.1 – UML Class Diagram of the Proposed Platform Architecture

5.2 Platform Architecture

In this section, we present the adopted architecture where the proposed approach is conceived. It is composed of three main parts: the *Native Platform* that we would like to secure, the *Remote Platform* sending the mobile agent and the *Trusted Third Party* (TTP). Figure 5.1 illustrates the structure of the adopted framework via an UML class diagram, showing the links between the different components.

— *Key_ Manager_ Agent*: It is an agent that makes use of the pseudo random number generator ISAAC+ [**Auma06**], to automatically generate a key of 256 bits for each file created in the environment. These keys and their corresponding hash obtained using SHA-3 [**Jaffar13**] are assembled in two separate lists.

— *Admin_ Agent*: It is responsible of the communication among all intern components and also with remote entities. Its tasks include mathematical operations to authenticate platforms, verify the keys and give access, in addition to managing the data of the environment and controlling their use.

— *Authenticator_ Agent*: It is charged with the authentication of any external entity claiming access to the platform. It collaborates with the TTP to perform a Key exchange integrating digital signature to prevent MITM attacks and generate a session key.

— *File_ Agents*: These agents contain the sensible files and data of the environment. They are protected and encrypted.

In the remote platform, we consider an "*Authenticator_ Agent*" with the same properties of that in the native platform, and a mobile agent containing two important information:

— *H(Key)*: The mobile agent may contain one or more specific keys for accessing one or more specific files. But this key is not given in clear, even the mobile agent did not know its value. The access keys are hashed using SHA-3, before they are provided along with an integer K to the mobile agent, via the TTP and through a secure channel.

— K: It is an integer referring to the number of concatenated access keys contained in the hash. It is useful for the native platform when receiving an authenticated mobile agent, to compute the arrangements of keys, that may construct the same hash.

In this context, it is necessary to notice that each mobile agent follows a preliminary itinerary defined by its own platform. Thus, the agent knows in advance the identities and locations (IP Address) of the platforms it will visit.

5.3 Authentication Process

To prevent vulnerabilities arising due to the unavailability of authentication, an agent charged with performing a mutual authentication is employed. This process inspired from [**Phan05**] is based on an enhanced Diffie-Hellman key Exchange protocol integrated with a digital signature DSA, which makes it well resistant to the Man-In-The-Middle Attacks. Figure 5.2 describes the authentication process that is composed of four phases:

1. *Key Request*: The remote platform communicates with the TTP in order to request the access keys to perform specific tasks. This request is then transferred to the native platform, which analyzes the required tasks and defines the concerned resources with their corresponding access keys. These latter are first concatenated and hashed using SHA3, then transferred to the TTP along with the integer K. The TTP in turn, relays this response to the requesting remote platform. We should mention that communications

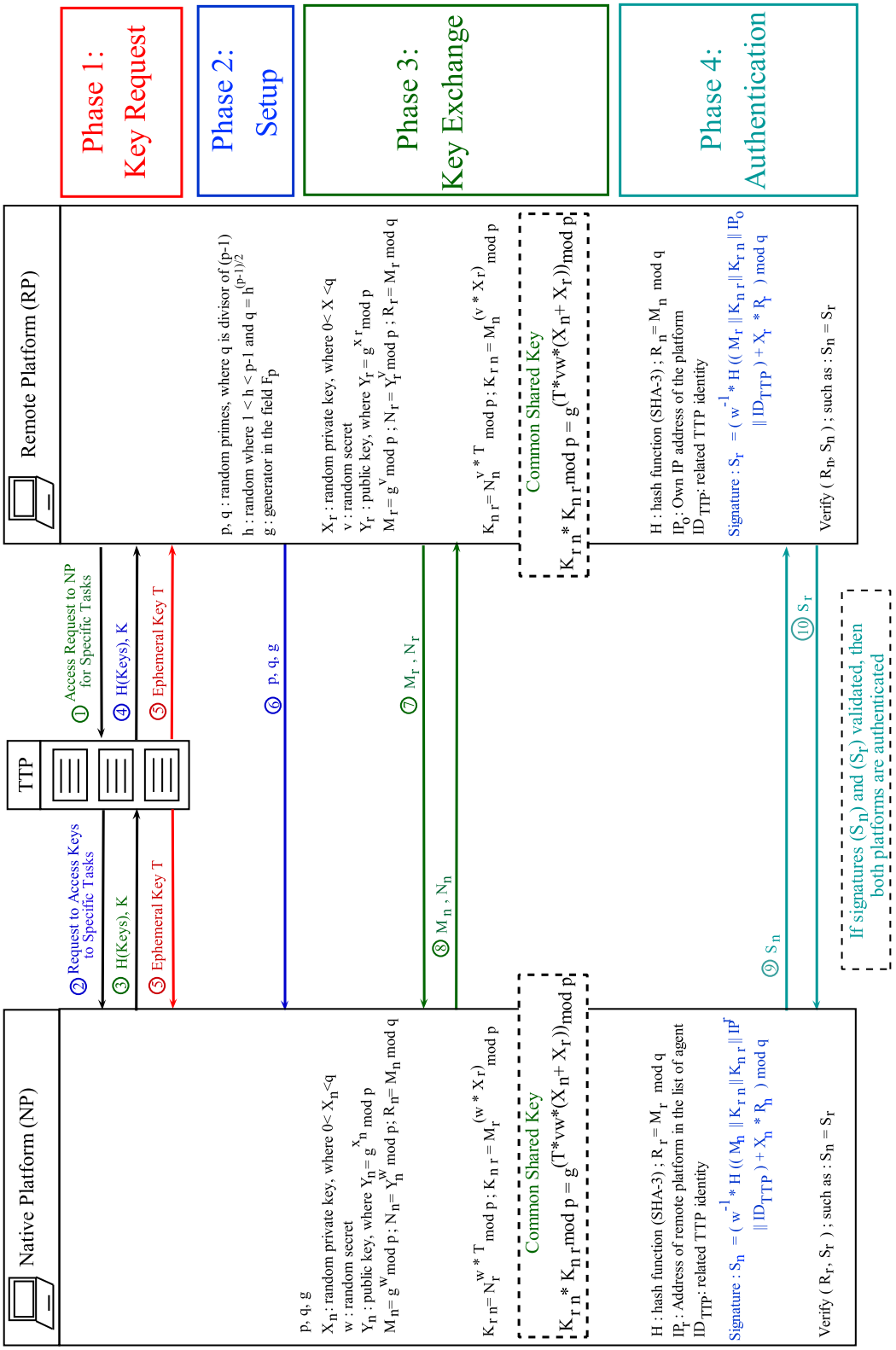

Figure 5.2 – Authentication Process between the Native and Remote Platforms Using TTP and Key Echange Mechanism

in this phase are carried out via a secure channel, through which, the TTP sends to the both platforms an ephemeral key (T) to be involved in the key exchange protocol.

2. *Setup*: Both interacting platforms agree on a set of basic parameters to perform computations. This set includes large primes (p, q and h) to define the rank of the finite field (F_p) to be used, and a generator (g) on this field to achieve exponentiation.
3. *Key Exchange*: In this phase, every platform generates randomly a private key (X_i), which is used to calculate the public key (Y_i) on the basis of setup parameters. Fixing the security of the protocol at this stage requires two main characteristics: the exchange of public key should not be in clear, and the number of passes must be minimized to prevent the intruder from collecting sensitive data. Thus, each platform chooses randomly an ephemeral secret (v and w), which is involved in the computation of the values (M_i and N_i) basing the public key. The overall exchange consists of two passes concluded with the generation of a session key, while its robustness is due to the hardness of the discrete logarithm problem.
4. *Authentication*: At this stage, each platform creates a digital signature through hashing a set of concatenated values. This set includes the parameters and calculated values of the previous phases, in addition to the relevant TTP identity and the appropriate IP address. Both signatures are then exchanged and verified to validate the authentication of communicating platforms.

5.4 Access Control of the Platform Resources

Access control limits the actions or operations that a legitimate user of a computer system can perform, which could lead to breach of security. Thus, in order to define an access control policy for our agent platform, we need to model our architecture as compounded of objects and subjects, such that the objects are the files stored and the subjects are the agents able to initiate actions or operations on objects.

For this approach, we choose to use DAC model for many reasons. First of all, it is a famous model widely used in information and operating system such as Linux. Also, it is flexible and appropriate for the systems where information sharing is important without compromising security. In our work, data are shared with the mobile agents and we need a flexible and light-weighted security policy that will not slow the exchanges between platforms and mobile agents.

The use of RBAC model in our case is not adequate, as it is based on roles directly associated with permissions. Thus, the problem is that the users having the same role possess necessarily the same privileges, which reduce the flexibility. To apply this model, we need absolutely to classify the agents into roles, where the agents having the same role must have the same access rights on the same files, which is not achievable in our modeling because each mobile agent has its specific access rights on specific files. In addition, there are no common rules among the agents (such as giving to the *AgentA* the right to access the files of *AgentB*, because any other agent having access to *AgentA* will get automatically the right to access the *AgentB*). This Access Control policy consolidates the security, as each agent is independent with its own access rights, and there is no common relations that might be a source of security attacks.

DAC model is designed such as the authorizations are controlled at the discretion of users, who are the controller or owner of some resources and assign the permissions to other users that may be strangers. Hence, in our platform the objects are represented by files containing data. The typical access rights defined here are: *Read*, *Write*, *Execute* and *Own*. The meaning of the first three access rights is self evident. *Ownership* is concerned about controlling who

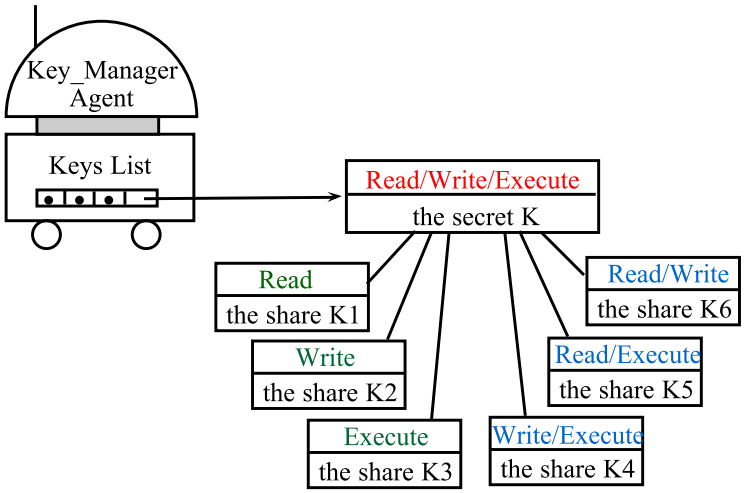

Figure 5.3 – Sharing the Access Rights Keys using Shamir's Threshold Scheme

can change the access permissions for the files or keys. Thus, the access right "*Own*" includes in reality the three others. This model is based on Access Matrix showed in Table 5.1, where each row refers to a subject (Agents), each column refers to an object (Files and keys). Each cell of the matrix specifies an access right, that is authorized for the subject in the row, and to the object in the column.

Table 5.1 – Access Matrix for the adopted Architecture

	File A	File B	...	Keys List	SessionKey
File_Agent A	O				
File_Agent B		O			
⋮			O		
Admin_Agent	W , R	W , R	W , R	R	R
Key_Manager_Agent				O	
Authenticator_Agent					O

O: Own R: Read W: Write

In order to specify each access right related to each file, we make use of Shamir's (n, t) Threshold Sharing Scheme presented in [**Blakl11**] to decompose the key of each file into sub-keys according to which access rights are granted. This scheme is designed such that each secret S in some field F is shared among n parties by creating a random polynomial $P \in F(x)$ of a degree t, such that $P(0) = S$. The i-th gets the share $(i, P(i))$. Given any $t + 1$ shares $P(x_0), ..., P(x_t)$, it is possible to recover $P(0)$(the secret S) using Lagrange Interpolation: $P(0) = \sum_{i=0}^{t} \lambda_i P(x_i)$, where: $\lambda_i = \prod_{j \neq i} \frac{x_j}{(x_j - x_i)}$.

Figure 5.3 defines $n = 6$ access rights extracted from the main three single ones, with a threshold $t = 4$ to reconstruct the original key, and with $l = 2$ the number of levels (level 1

with one access right and level 2 with two concatenated access rights). This sharing structure may be extended such as: if s is the number of main single access rights, then the number of shares is $n = \sum_{i=1}^{s-1} C_s^i$, while the number of levels is $s-1$. Let us consider a simulation scenario where a mobile agent is carrying two concatenated keys $(H(Key1 + Key2), k = 2)$ to access two files with specific access rights. We suppose that the native platform contains four files, and the mutual authentication is successfully performed, Figure 5.4 describes the processing steps, in normal conditions, of the proposed access control policy.

— *Step 1*: The mobile agent is encrypted using the session key and AES algorithm [**Rober12**], then it migrates to the native platform. We choose AES-256 due to its very convenient use as it consumes little memory, for its reduced complexity and for its easy implementation. Once arriving to the native platform, the mobile agent is decrypted by the relevant "Admin_Agent" to get the credentials: $H(Key1 + Key2) and k = 2$. At this moment, there is no risk that the mobile agent will be duplicated because all the system is fault tolerant. When an unexpected failure happened, the mobile agent will resume its execution and recover the last saved state. Besides, the mobility from one platform to other is transactional and each mobile agent can only be executed in one node at a time.

— *Step 2*: In possession of mobile agent credentials, and knowing that for each file in the platform there is 7 keys specifying 6 access rights and a global secret key, the "Admin_Agent" has to construct a list of arrangements without repetition of two elements among the 28 keys (7 keys for 4 files). The order of these arrangements is very important due to the use of hash function, where: $H(Key1 + Key2) \neq H(Key2 + Key1)$. It is worth to mention, that the two keys are belonging to different files, which means that the arrangements of keys for the same file are eliminated. Thus, the final list contains effectively $A_{28}^2 - 4 \times A_7^2 = 588$ possibilities, and it is sent to the "Key_Manager_Agent" along with the hash of the two concatenated keys .

— *Step 3*: Once receiving the list of key-arrangements, the "Key_Manager_Agent" performs a restricted search for the corresponding hash. If this later is found, then the files subject of this arrangement are recognized with the corresponding access rights. Thereafter, the "Key_Manager_Agent" decrypts the relevant files, adjusts them according to the access rights assigned, and notifies the "Admin_Agent" to allow the mobile agent to execute its tasks on the named files. Else, if the hash is not found, then an acquittal is sent to the "Admin_Agent" in order to terminate the mobile agent.

5.5 Security Analysis

In this section, the resistance of the proposed solution against some well-known attacks is investigated.

Conspiracy Attack

Consider 3 mobile agents belonging to 3 different remote platforms, can these platforms work together to collect correlated parameters of the native platform? If the 3 mobile agents claim access to the same file with different modes (Write, Read, Execute), can they collaborate within the native platform to reach a higher access right than they possess?

First of all, our contribution benefits from the security and ability of Shamir's Threshold scheme to prevent this attack. Therefore, our security policy is fine-grained according because

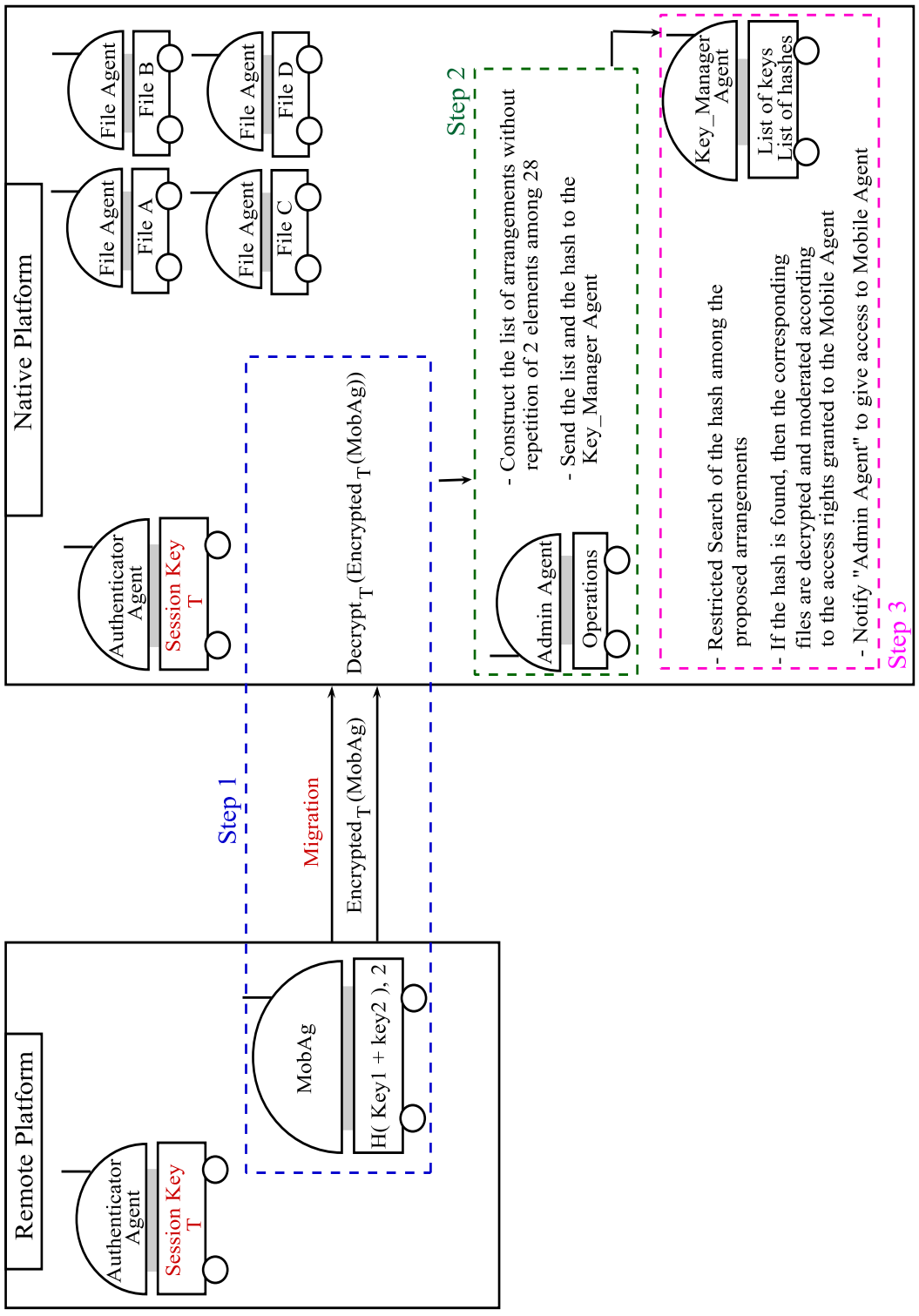

Figure 5.4 – The Process of the Proposed Access Control Policy for the Native Platform

the "Admin_Agent", as the mediator among agents, performs all necessary computations without disclosing the private parameters of the environment. Besides, the received mobile agents are transporting hashes without knowing their real values, which means they need absolutely to resolve the barrier created by exponential operation, large prime number application and one way hash function. Thus, this attack will fail.

MITM Attack

The authentication process adopted for our proposed solution proves its resistance against the Man-In-The-Middle attack. This is due to the fixation of the integrated DHKE-DSA using ephemeral keys, such as the computations of the two session keys K_rn and K_nr depend on the ephemeral keys r and n as well as T shared by the TTP via a secure channel. It provides two important properties. The first one is perfect forward secrecy, because in the long-term, if a private key of any party is exposed, previous session keys cannot be computed since the ephemeral keys of that session are unknown. The second one is key freshness, as every session key is a function of ephemeral keys, which means no party can predetermine the session key's value since he would not know the other party's ephemeral key.

Reverse Attack

In this attack, when two platforms are belonging the same level of hierarchy, then each one of them can use its public parameters and relationships to access or modify the internal information of the other platform. In our approach, our native platform is hierarchically independent and only relies to the TTP, that manages its remote communication according to specific constraints. Thus, in case that there exists a relationship between our platform and another one, this latter cannot access internal information of the first one.

In our approach, it is very difficult for the attacker to know the superkey of the native platform, due to the TTP that interacts through highly secure channel. Moreover, the use of the one way hash function provides the non-reversibility property, that makes the derivation of the input data from output data impossible. In addition, if that attacker intends to intercept the messages from the native platform, it must absolutely recover the session key, and then it will be faced to the challenge of solving the discrete logarithm problem (DLP) and the complex exponential computations with large prime numbers.

5.6 Performance Analysis

In this section, we present the experimental investigations to prove the feasibility, effectiveness and the detection capacity of the proposed approach. Our evaluation is based on three factors: time performance and detection performance. Thus, we have designed and implemented the proposed architecture basing real machines with Core i7 processor at 2.7 GHz, and 4 Go of RAM. All machines are equipped with JADE agent platform [**Belli01**] in its 4.3.3 version.

Time Cost

To compute the time cost of our solution, we need to compute the time of each operation performed during the round trip of the mobile agent to access the native platform data. These calculated times include the following operations:

— T_{RAK}: the request of the access keys,

— T_{KEA}: the key exchange with the authentication,
— T_{EDMA}: the encryption and decryption of the mobile agent,
— T_{Mig}: the migration of the agent,
— T_{CAL}: the construction of the arrangement list,
— T_{HS}: the hash search,
— T_{DMF}: the decryption with moderation of the corresponding files,
— T_{MAE}: the mobile agent execution,
— $T_{EMA} + T_{Return}$: the return of the encrypted mobile agent to its own platform,

Let T_{total} be the execution time for the previously mentioned operations:

$$T_{Total} = T_{RAK} + T_{KEA} + T_{EDMA} + T_{Mig} + T_{CAL} + T_{HS} + T_{DMF} + T_{MAE} + T_{EMA} + T_{Return}$$

Knowing that T_{Mig} and T_{Return} are approximately equal, and the time to encrypt an agent is the same as to decrypt it, then:

$$T_{Total} = T_{AKR} + T_{KEA} + \frac{3}{2}T_{EDMA} + 2T_{Mig} + T_{CAL} + T_{HS} + T_{DMF} + T_{MAE} \quad (5.1)$$

Table 5.2 – Time of the different operations performed during the process of the proposed solution for one mobile agent with K=2

Operations	**Time (ms)**	
T_{AKR}	33,7	
T_{KEA}	8,6	
T_{EDMA}	30	
T_{Mig}	46,6	$T_{total}(K=2)$ **=301,1 ms**
T_{CAL}	3,4	
T_{HS}	2,1	
T_{DMF}	62,3	
T_{MAE}	23,8	

Table 5.2 presents the measured times of the operations mentioned for the previous scenario, where a remote platform sends a mobile agent with two concatenated keys to get access to two files among four ones, and the total time is calculated according to Equation 5.1. These time costs has been measured for the system setup of two real platforms, where a native platform has interacted with a remote one for 100 times. Likewise, the use of JADE as agent framework adopting Java programming language, allows us to take benefit from the services of its security providers and its Java cryptography extension (JCE). The process is reiterated with two files of 500 Kbytes and 1000 Kbytes, then the cost time of each operation is calculated as the average of its 100 iteration times.

We find it interesting to extend our experiments, such that we first launch a baseline test without security operations, and then a test of our proposed solution. This will allow us to know the overhead of the security added to the system, as well as to show its effectiveness and its reliability to detect malicious entities. Both experiments are performed through the

configuration of a system architecture composed of six real platforms, where a native platform hosting 20 files of different sizes is interacting with five remote platforms. In each test, every remote platform sends 20 mobile agents to get different access rights to specific files. Among the total 100 mobile agents there are 40 which are malicious. It should be noticed that for both tests, the process is launched 100 times and the average time value is kept.

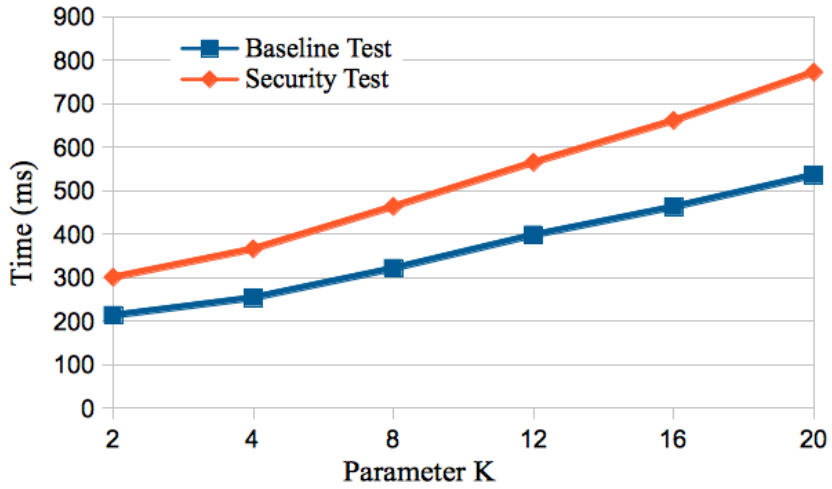

Figure 5.5 – Time cost of Baseline Test and Security Test face to the increase of parameter K

Figure 5.5 shows the time costs of the baseline test and the security test, regarding to different and growing values of the parameter (K). Besides, Figure 5.6 gives the total time that each test spent to interact with increasing number of mobile agents. The obtained results show the ability of our solution to scale in front of large conditions, namely the expanded number of mobile agents requesting to access platform services, the set up of growing databases and especially the advantage to continue performing in non-connected areas. Moreover, the overhead of security added is about 40% of the the overall cost, which is credible and highly beneficial for the platform to counter external vulnerabilities.

For comparative purposes, we are referred to the method of Leila Ismail [**Ismail08**], where the authentication and access control to the system resources are controlled by the mobile agent platform. This allows transparent and dynamic way to grant access rights among cooperating agents through access control capabilities, while the authentication mechanism is ensured using a Certification Authority (CA). Figure 5.7 presents the measured time costs for a round-trip of this method compared to the our, for different file sizes: 500Kbytes, 1000Kbytes, 1500Kbytes and 2000Kbytes respectively. It demonstrates the effectiveness of our solution with a security overhead of 40% versus 67% for the other one.

Detection Capacity

It was worthy at this stage to evaluate the capacity of our solution to detect maliciousness related mainly to unauthorized access attacks. Figure 5.8 compares the performance of both proposed tests to detect harmful mobile agents. When the Baseline Test was able to detect only 10 malicious agents among 40, which represents 25% of the total malicious agents number, our scheme was able to detect the forty agents trying to get access without being authorized, which

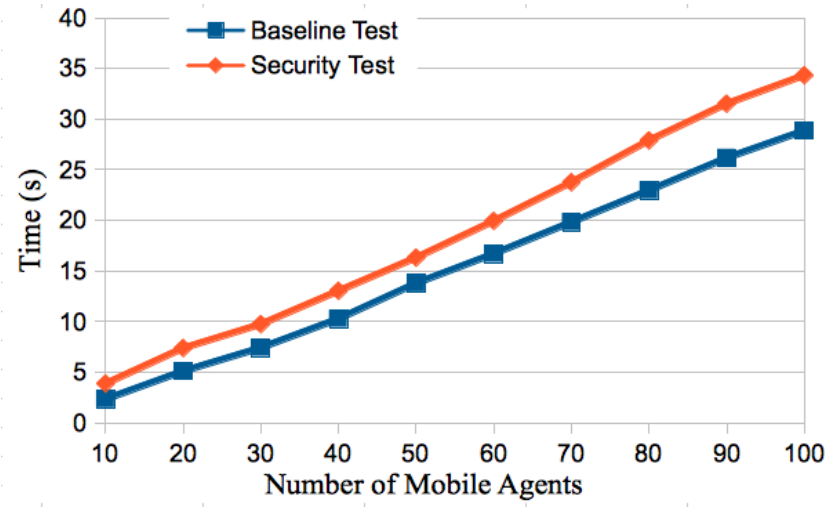

Figure 5.6 – Time cost of Baseline Test and Security Test face to the increase of interacting mobile agents

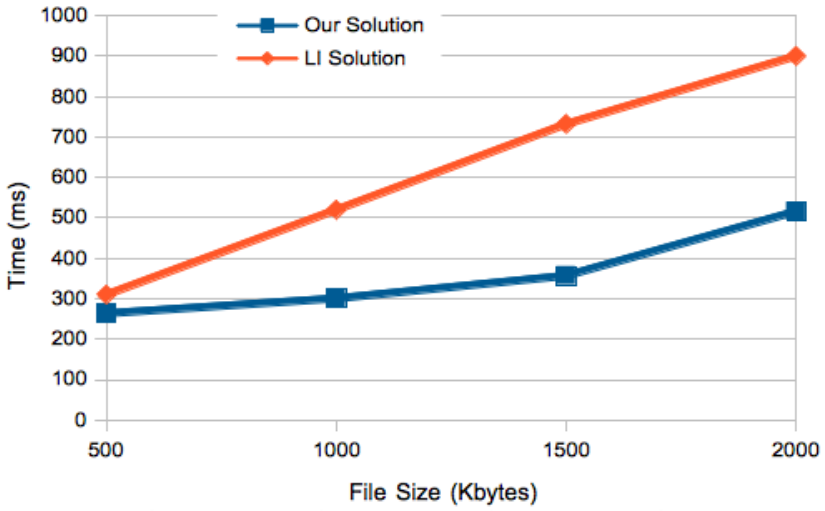

Figure 5.7 – Comparison of Time Performance of our Solution versus to the Solution in [**Ismail08**]

means 100% of the total malicious agents number.

5.7 Conclusion

In this chapter, we have dealt with the security issues that a hosting platform may face when receiving mobile agents. Among the serious concerns in this context are the unauthorized access attacks, that could harm the entire system and be extended to external entities. For that

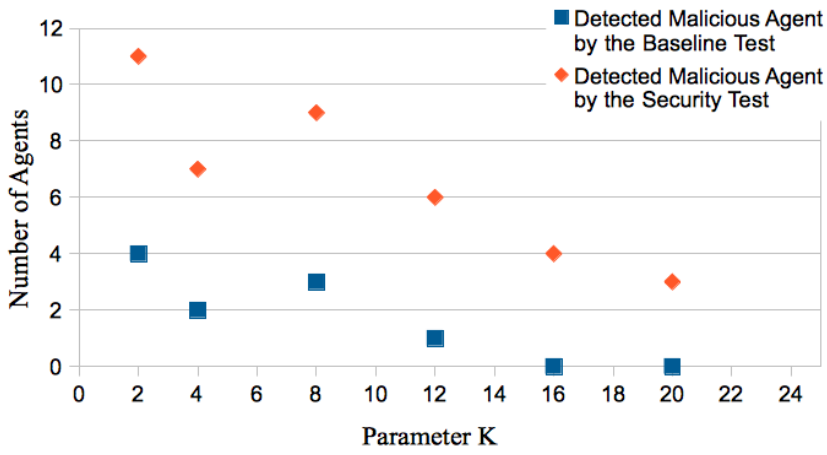

Figure 5.8 – Comparison of the Malicious Agent Detection Performance of the Baseline Test and the Security Test

purpose, we have associated access control and cryptographic mechanisms in a new combination, that shows its effectiveness to fill the majority of security needs: the authentication using DHKE-DSA, the confidentiality using AES encryption by a session key, the integrity through SHA-3 hush function, and access control along with availability management through using DAC and Shamir's threshold scheme. Our solution proves its scalability and reliability in detection of malicious mobile Agents.

Chapter 6

Application: Cloud Security using Secure Mobile Agent

Contents

Cloud Computing is a rapidly expanding paradigm that brings a revolution in IT world through using the Internet services. These services that consist of applications and databases deployed in large centralized data centers, are delivered to end users on demand rather than being maintained in a large and expensive IT infrastructure. Cloud computing was defined by the NIST [**Mell11**] as an emerging computing approach enabling ubiquitous, convenient and on-demand network access to shared resources (e.g., data, servers, applications, and services), that can be rapidly provisioned and released with minimal management effort or service provider interaction. Available and accessible data/services, reliable and flexible computations on demand, in addition to the wide storage capacities associated with high quality of services, are among the numerous benefits of the cloud that attract customers and make it a viable commercial option, particularly for small companies and startups which will potentially reduce their costs when paying only for what they really use.

Even though Cloud Computing shows a significant widespread according to its recent design of IT hardware, the security of clouds has become an important issue. With distributed and open structure mainly based on resource virtualization, global replication and migration, cloud computing increasingly attracts potential intruders. In 2011 [**Galan11**], the Amazons Elastic Computer Cloud service is used by a hacker to attack Sonys online entertainment systems, by

registering and opening an Amazon account and using it anonymously. Such attack compromised more than 100 million customer accounts, the largest data breach in the U.S. One of the most common requirements for cloud security is Intrusion Detection and Prevention System (IDPS), which can be efficient to early detect malicious entities, track their untrustworthy behaviors and prevent serious damages to the systems. An IDPS was also defined by the NIST [**Scarf07**] as a software or hardware device, that has all the capabilities of an intrusion detection system (IDS) to potentially identify an attack and notify appropriate personnel immediately, and can also attempt to stop possible threats or at least prevent them from succeeding, so that they can be contained.

In this chapter, a robust IDPS relying on mobile agent technology is proposed, in order to ensure a safe interaction model where the communication between the cloud provider and its related cloud storage servers is secure and reliable. The use of this technology allows us to benefit from the autonomy, pro-activity, mobility and flexibility of its entities, as well as from the security aspects provided throught highly secure and complex mechanisms, which provides the cloud with new intelligent and reliable solutions. According to [**Talia11**], the qualities offered by the mobile agents make them well suitable for user access management through a negotiation process, as well as for the automation, composition and trading of the resources and services, which make clouds smarter and more efficient in their interactions and processing.

Our approach begins once the cloud provider receives requests from one or multiple cloud users. Then, it creates a mobile agent with specified constraints, and which will be charged for the migration across the cloud storage servers to perform computations, collect data, benefit from services, and then it will return back to the cloud provider with final results. Along its round-trip, the mobile agent adopts two major mechanisms combined with cryptographic primitives (asymmetric encryption, digital signature and hash function) to ensure confidentiality and integrity of data associated. The first mechanism consists of generating cryptographic traces, where all behaviors, actions and computations performed either by the agent or the hosting server are recorded, so that any anomaly can be easily detected through verifying and analyzing these traces. The second is a prevention mechanism based on revocation technique, where a trust threshold is assigned to each server to define its degree of maliciousness, according to many factors and proofs. Once this threshold is reached, the named server is added to the black database of malicious servers hosted by a specified authority.

The reminder of this chapter is organized as follows. Section 2 defines the context of this application, where a statement of the security problems in cloud computing, especially those between the Cloud Provider and the storage servers, and a brief review on the related works are provided. In Section 3 we describe the proposed cloud-IDPS. Then, an evaluation of the application performances using ClouSim tool is exposed in Section 4, such that the obtained results are compared to a basic mathematical model, in terms of security, reliability and efficiency. Finally, further discussion and perspectives are mooted in conclusion.

6.1 Application Context

This section exposes the context and the motivation of this work. Thus, we begin by introducing the security problem in the cloud computing. Then, we locate our application compared to other related works investigating the security issue of the cloud.

6.1.1 Problem Statement

While cloud computing makes a large set of advantages more appealing than ever, it also brings new and challenging security threats to the outsourced data. This is relatively due to the virtualization concept and the physical absence of data and machines, which makes the resources and services within the cloud architecture not fully secure neither well monitored and managed.

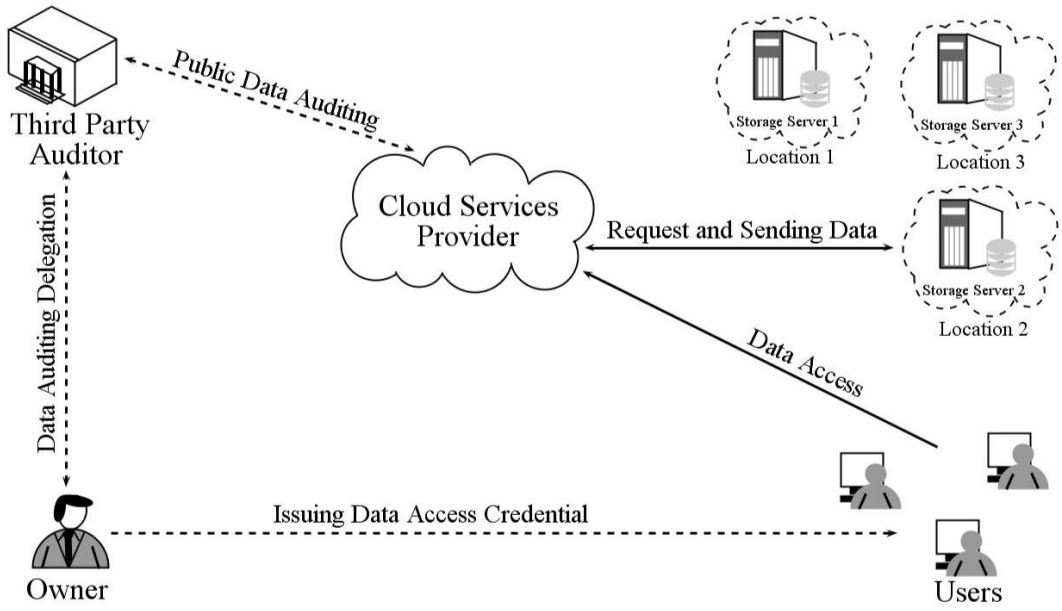

Figure 6.1 – The Generic Architecture of Cloud Computing

We begin with an architecture description of cloud data storage services illustrated in Figure 6.1. This latter consists of four different entities: data owner, cloud user (CU), cloud service provider (CSP) controlling and monitoring numerous storage servers (SS), and TPA that has the capabilities to assess cloud storage security on behalf of a data owner. When a CU submits storage or computation service requests, the CSP charges its administration server to spread the request among its different SS located on different geographical areas. Among the essential features that Cloud Computing provides are defined:

— *Broad Network Access*: Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).

— *Resource Pooling*: The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or data-center). Examples of resources include storage, processing, memory, and network bandwidth.

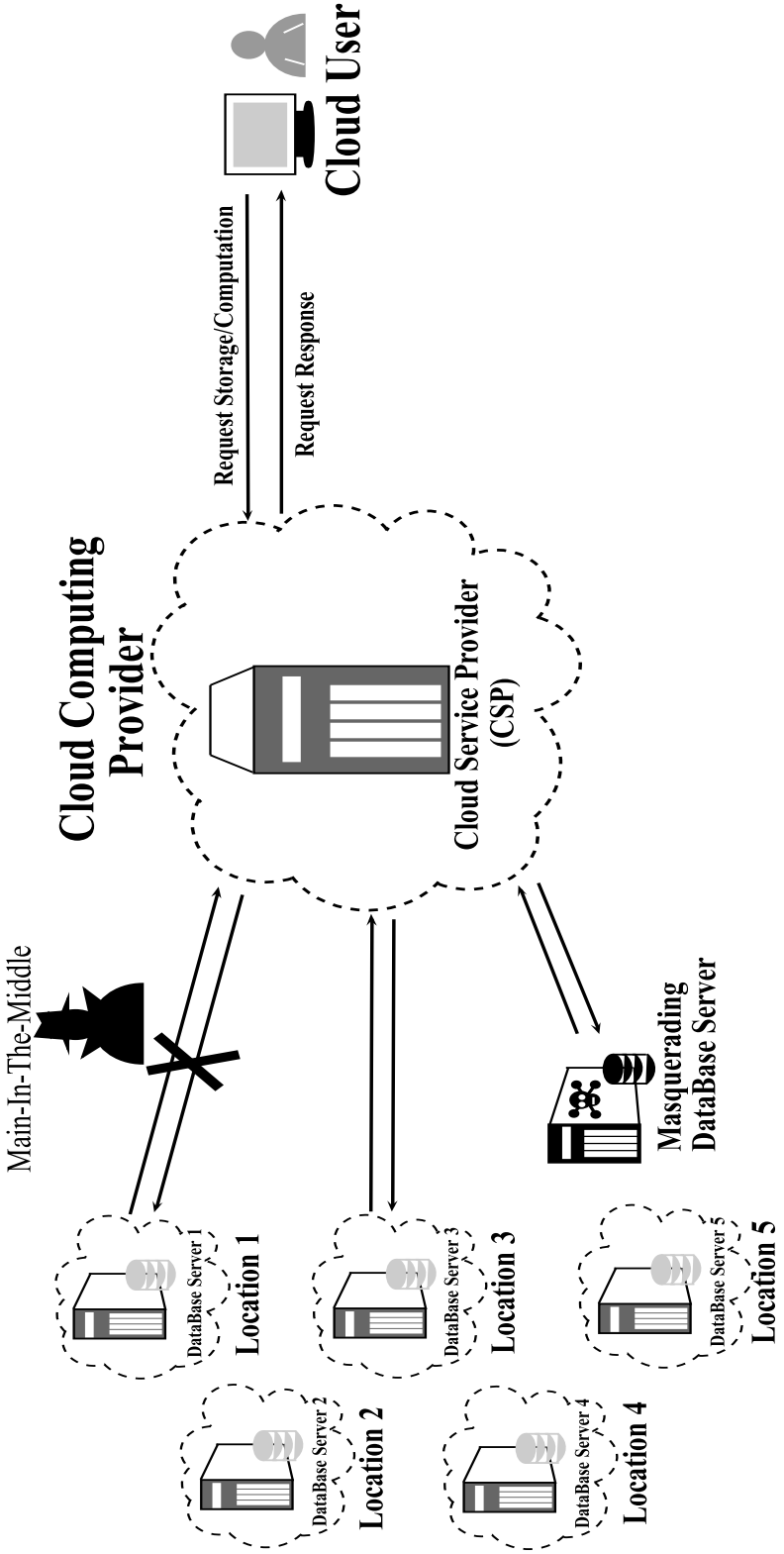

Figure 6.2 – Security issues in Cloud Computing architecture

These two features are highly vulnerable to several attacks (such as: Man-In-The-Middle attack and Masquerading attack) and raise many security problems as shown in Figure 6.2. In this context, multiple scenarios are considered:

— An adversary A could corrupt one or multiple SSs and control them to launch various cheating attacks.

— An adversary could intercept the communication between the CPU and the cloud servers, and modify the exchanged data to compromise their confidentiality and integrity.

— An adversary may compromise CUs privacy by leaking their confidential data to others.

Within the scope of this article, we focus on how to ensure secure cloud data storage services. We consider both malicious outsiders and a semi-trusted storage server SS as potential adversaries interrupting cloud data storage services. Malicious outsiders can be economically motivated, and have the capability to attack cloud storage servers and subsequently pollute or delete owner's data while remaining undetected. The CS is semi-trusted in the sense that most of the time it behaves properly and does not deviate from the prescribed protocol execution. However, for its own benefit the CS might neglect to keep or deliberately delete rarely accessed data files that belong to ordinary cloud owners. Moreover, the CS may decide to hide the data corruptions caused by server hacks or Byzantine failures to maintain its reputation.

6.1.2 Related Works

Many efforts have been devoted to investigate the security issue of Cloud Computing. Indeed, several solutions have been proposed in the literature, where IDPSs are supplied as very important and invaluable tools. Towards this, [**Tupak11**] proposed a Cloud-IDPS based on a virtual machine monitor, called hypervisor to protect the system from different types of attacks in the infrastructure layer (IaaS). However, it has not provided a prevention solution face to high severe attacks over the system.

[**Kholi12**] propose a fully distributed system P2P architecture with no central manager coordinator. Their system adopts hybrid detection techniques using network and host based audit data, which provides a flexible, robust and elastic solution for cloud computing, but not sufficient to detect large scale attacks due to the absence of a central correlation master.

Recently, Autonomic Computing drew researchers attention for CIDPS with minimal human intervention. [**Smith10**] proposed an autonomic mechanism for anomaly detection in a cloud computing environment, with uniform format analysis and size reduction for data, as well as learnt how to detect the nodes which have abnormal behavior and act differently from others in an unsupervised mode.

Ontology features are also used, such as uCLAVS the detection malware based on intrusion ontology representation proposed by [**Marti10**]. Their model provides a multi-engine based files analysis service instead of running complex software on every host to analyze the files individually.

[**Dastj10**] proposed an application of mobile agents in IDPS to provide flexible, scalable, and a cost effective system for the cloud environment. However, the inefficient sharing of knowledge among the mobile agents makes the robustness of the system not supported and the scalability not ensured.

6.2 Proposed Cloud-IDPS

In this section, a thorough description of the proposed Cloud-IDPS is provided, which consists basically of two main parts. The first one concerns our contribution to elaborate a

robust detection mechanism, based on cryptographic traces produced by autonomous agents during their mobility across cloud servers. The second part presents our prevention policy that punishes malicious entities through a server revocation technique based on trust threshold. The detection and prevention in our proposed system are complementary and dependent, as they are basically reliant to the same sensible parameters in the cryptographic traces obtained through the trip of the mobile agent, to perform the tasks that the CU requests. First, let us consider Figure 6.3 as an explicit scheme describing functional layering in an IDPS. Generally, there are three essential security functions that an IDPS should serve:

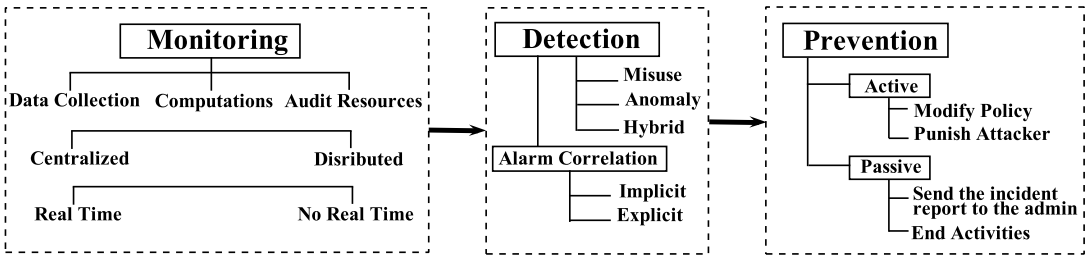

Figure 6.3 – IDPS Functional layering

1. **Monitoring and Detection**: as discussed in Chapter 4, the monitoring may concern network traffic to analyze particular segments and devices, or dynamic behaviors stating which actions are performed on which resources, or events occurring on specific applications and which affect the overall performance. The detection is occurred through analyzing the collected data or the stated behaviors. It makes use of three main categories of detection methods: Misuse, Anomaly or Hybrid. Thus, when an attack is detected, the system generates one or multiple alarms which are clustered into incidents (Correlation). This could be achieved implicitly through data-mining techniques, or explicitly using patterns based logical and temporal constraints languages.
2. **Prevention**: in order to prevent the intrusions, the IDPS responds actively by modifying its access control policy temporarily or punishing the attacker through changing its state or routing him elsewhere. Passive responses are also welcomed, where attacker activities are blocked and terminated through ending connection sessions.

6.2.1 Cryptographic Traces for Intrusion Detection

In this section, we describe the proposed detection method, which is mainly based on the execution tracing associated to the mobile agents. Thanks to its numerous features, mobility of agents makes interactions with cloud servers more flexible and efficient through minimizing the number of messages exchanged, which help in reducing network traffic. As illustrated in Figure 6.4, in a normal scenario, the CSP receives multiple requests for computation or storage services from different CUs. According to this, a mobile agent is created and attributed a list of the IP addresses to visit and the tasks to perform on each one. Then, the mobile agent analyzes the given positions to follow the shortest path, and moves among the specified cloud servers to execute its tasks, such that the obtained results are securely accumulated until it returns backs to the CSP.

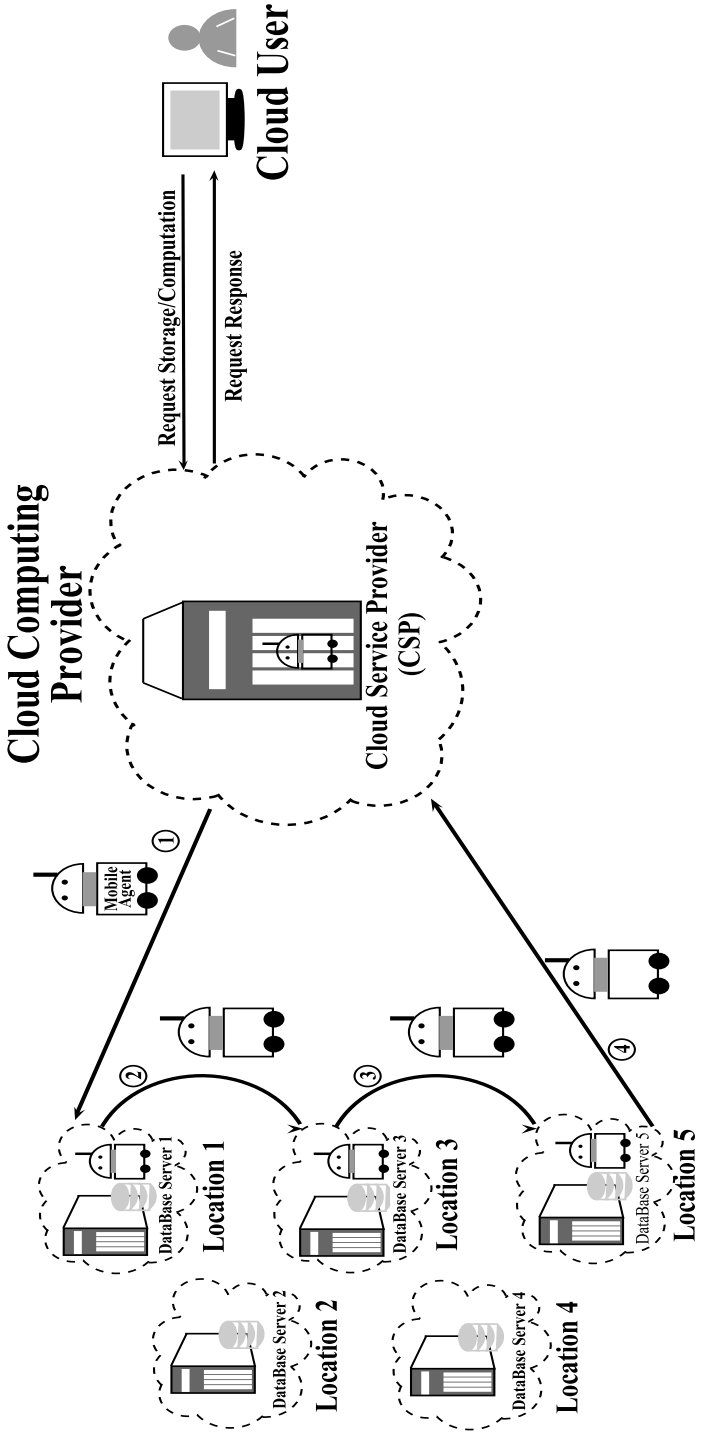

Figure 6.4 – The use of mobile agents in the interactions of the cloud computing

Before migration, the agent is also assigned some essential credentials given in Table 6.1, to authenticate the visited cloud servers according to their IP addresses. Thus, we make use of an enhanced version of Diffie-Hellman key exchange protocol inspired from **[Phan05]**, where the digital signature algorithm is integrated to fix security issues related to Man-in-the-Middle attacks. Figure 6.5 illustrates the authentication process, that generates a common session key at the end. This process employs the cryptographic generator ISAAC+ to produce randoms and the hash function SHA-3 to elaborate signatures.

Table 6.1 – Authentication Credentials assigned to the mobile agent by the CSP

Credential	**Description**
p, t	random odd primes, where $t < p-1$ and $q = t^{\frac{(p-1)}{2}}$
g	generator of the field F_p
X_{MA}	random private key, with $1 < X_{MA} < q$
PK_{CSP}	public key of the CSP
$H(.)$	256-bits hash function (SHA-3)

The use of ephemeral secrets a and b chosen by the both sides provides two important properties. The first one is *forward secrecy* that prevents the disclosure of any of the previous session keys even if the long-term private key of any party is exposed. The second one is *key freshness* as neither of the authenticating parties can predetermine the value of the session key, since he would not know the ephemeral secret of the other party. In addition, the use of the CSP's public key in computations facilitates traceability and saves time and energy of producing a public key for the agent at each cloud sever visited. Thereby, the 256-bits length session key (SK_n) obtained at the cloud server (n) will be used to compute the cryptographic trace on that platform.

Being inspired from the tracing technique in **[Vigna98]**, a trace is associated to the execution of the mobile agent on each visited server. It is noted $T =< ui, S >$ and contains the signature of the executed black statements, in addition to a unique identifier of the trace produced randomly using the Java package java.util.UUID. This process allows the owner of the agent to verify its execution and depict supposed behaviors or claimed operations performed by malicious intruders aiming at tampering the agent. Indeed, the temporal correctness of agent execution is a crucial point to guarantee that the connected cloud servers work in synchronized timing. Thus, we introduce Network Time Protocol (NTP) **[Mills10]** to enforce clocks of the different cloud servers under control of the CSP to perform in sync with him. This makes the control and matching of log entries more straightforward when an event occurred across multiple servers.

A cryptographic trace is generated as indicated in the Java function illustrated in Figure 6.6. It contains a generic DSA signature performed using the public key of the CSP, which makes this later the only one able to decrypt it by means of its private key. In addition to the unique identifier (ui), the identities (ID_s: sender, $host$: current host, ID_next: intended host), the session key ($SK\_S(i)$), the timestamp (ts) and the required task ($Task$), the generic signature includes two nested fields:

— A symmetric encryption, using AES and the 256-bits $SK_S(i)$, of black statements and results obtained through execution of requested task.

Mobile Agent (MA)		**Cloud Server ($\mathbf{CS_n}$)**
a: random integer	$\xrightarrow{p,q,g,H}$	
		X_{CS_n}: random private key $< q$
		Y_{CS_n}: public key, with:
		$Y_{CS_n} = g^{X_{CS_n}} modp$
		b: random integer
$V_{MA} = g^a modp$,		
$W_{MA} = PK^a_{CSP} modp$		
$R_{MA} = V_{MA} modq$	$\xrightarrow{V_{MA},W_{MA}}$	
		$V_{CS_n} = g^b modp$,
		$W_{CS_n} = Y^b_{CS_n} modp$
	$\xleftarrow{V_{CS_n},W_{CS_n}}$	$R_{CS_n} = V_{CS_n} modq$
$K_{ab} = W^a_{CS_n} modp$,		$K_{ba} = W^b_{MA} modp$,
$K_{ba} = V^{a \times X_{MA}}_{CS_n} modp$		$K_{ab} = V^{b \times X_{CS_n}}_{MA} modp$
IP_h: address in the Host list		IP_{CS_n}: address of the platform
		Signature:
		$S_{CS_n} = b^{-1} \times H((V_{CS_n} \parallel K_{ba} \parallel K_{ab}$
	$\xleftarrow{S_{CS_n}}$	$\parallel IP_{CS_n}) + X_{CS_n} \times R_{CS_n}) modq$
Verify: $S_{CS_n} == S_{MA}$, where:		
$S_{MA} = a^{-1} \times H((V_{MA} \parallel K_{ab} \parallel K_{ba}$		
$\parallel IP_h) + X_{MA} \times R_{MA}) modq$		

Common Shared Key

$$\mathbf{SK_n = Kab \times K_{ba} modp = g^{vw \times (X_{MA} + X_{CS_n})} modp}$$

Figure 6.5 – Authentication Process between the mobile agent (MA) and the cloud server (CS)

— The DSA signature, using $SK\_S(i)$, of the agent's identity and its sender, as well as a SHA-3 hash of the timestamp, black statements and the results of execution.

In order to ensure a chaining mechanism among the traces produced during the trip of the mobile agent, the trace that has been generated on the previous cloud server (the agent's sender) is also enclosed in the generic signature, while its hash is joined to the nested signature. This only concerns cloud servers starting from order $i = 2$, as for the first visited server no previous trace is provided by the sender that is the CSP.

Once returning back to the CSP, the mobile agent presents the final results along with the collected traces. Then, a verification of each trace $T(S(i))$ is performed following the steps below:

1. Using its secret key, the CSP decrypts the generic signature, verifies the identities in-

```
public Trace<ui,S> Generate_CryptoTrace( Agent A, CloudServer S(i)){
  ID_s = A.getSender();
  ID_ma = A.getAgentIdentity();
  host = S(i).getIdentity();
  crd = A.getCredentials;
  PK_csp = crd.getCSPkey();
  ID_next = crd.getNextHost(S(i));
  SK_S(i) = this.getSessionKey();
  Task= A.getRequestTask();
  BS = class.execute(A).getBlackStatement();
  RS= class.execute(A).getResults();
  // current timestamp
  java.sql.Date ts = new java.sql.Timestamp(Calendar.getInstance().getTime().getTime());
  ui = UUID.randomUUID(); // unique identifier
  Signature dsa1 = Signature.getInstance("SHA256withDSA", "SUN"); dsa1.initSign(PK_csp);
  Signature dsa2 = Signature.getInstance("SHA256withDSA", "SUN"); dsa2.initSign(SK_S(i));
  if (i==1)
  T(S(i))= <ui, dsa1.sign(dsa1.update(ui, SK_S(i), ID_s, host, ID_next, ts, Task,
           AES_Encrypt( SK_S(i), (BS,RS)), dsa2.sign(dsa2.update(ID_s, ID_ma,
           Hash(ts, BS, RS) ) ) ) )>;
   else
  T(S(i))= <ui, dsa1.sign(dsa1.update( T(S(i−1)), ui, SK_S(i), ID_s, host, ID_next, ts, Task,
           AES_Encrypt( SK_S(i), (BS,RS)), dsa2.sign(dsa2.update(ID_s, ID_ma,
           Hash(ts, BS, RS), Hash (T(S(i−1))) ) ) ) )>;
 return T(S(i));
}
```

Figure 6.6 – Java pseudo-code of the cryptographic trace generation

volved and checks the freshness of the timestamps (ts). Then, decrypts the cipher to get the black statements and results in clear.

2. A SHA-3 hash of the previous trace provided by the mobile agent is calculated: $h1 = Hash(T(S(i-1)))$.

3. Using the current session key $SK\_S(i)$, the CSP decrypts the nested signature and verifies the identity of the agent and its sender.

4. The CSP computes its own hash of the timestamp along with the black statements and results, that are provided in the cipher. Then, this hash is compared to that contained in the third field of the nested signature.

5. The CSP computes the hash of the given trace in the first field of the generic signature: $h2 = Hash(T(S(i-1)))$. Then $h1$ and $h2$ are compared to the hash in the last field of the nested signature.

Once the verification is successfully carried out, the cloud server is classified as trustworthy for this session, else it is considered suspicious and the CSP proceeds to its prevention policy.

6.2.2 Revocation-based Trust Threshold for Intrusion Prevention

As a matter of fact, the prevention policy we adopt for our system consists in a double faced method: ending the activity of the malicious server in that session, and punishing it through decreasing its level of trust.

When the CSP verifies the trace of the mobile agent's execution on a server $S(i)$, and finds that one or more comparisons do not match, then this server is qualified as suspicious. At this stage, the CSP sends a message to the named server asking it for forwarding the specified trace. The core of this message and its reply provided by $S(i)$ is shown in Figure 6.7, such that the identity of each one is involved as identifier for its message.

```
From CSP to S(i):
M_(csp-to-si)=Sign_(sk_(si)) ( Hash(ID_(si)), Forward-Trace())

From S(i) to CSP:
M_(si-to-csp)=Sign_(pk_(csp)) (Hash(ID_csp), Hash(ID_(si)),
                                    M_(csp-to-si), T(S(i)))
```

Figure 6.7 – The core of the message and reply for Forward-Trace() transaction

Once the CSP receives the reply for its request and decrypts the involved signature using its private key, it proceeds to the verifications as indicated in Figure 6.8.

1. Checks the hash containing its identity ID_csp, then verifies that the value $Hash(ID\_(si))$ is the same in both messages $M\_(csp-to-si)$ and $M\_(si-to-csp)$.
2. Decrypts the nested $M\_(csp-to-si)$ using the session key and verifies that it contains the forward request.
3. Computes the hash of the trace provided $Hash(T(S(i)))$ and verifies that it matches with the hash given by the server $S(i+1)$.

It is sufficient that one of these verifications does not match, so that the CSP can start its revocation protocol in concordance with a Trust Authority (TA). Otherwise, it charges the same mobile agent with moving to the server $S(i)$ to re-execute the same tasks, and then verifies that the execution agrees with the forwarded trace. If the traces match, then the server is qualified as honest, else the revocation protocol is triggered.

The revocation protocol has two instances according to the trust level of the cloud server. A trust level (TL) is represented by a decimal number between 0 and 1 and it is allocated by the Trust Authority (TA), which is the only one who possesses the right to modify it. According to this (TL), the revocation may be conditional or permanent, with regard to a threshold (d) decided by the TA for each cloud server registered in its database.

The conditional revocation protocol is processed when the cloud server has a TL that did not reach yet the threshold (d), and it is described as follows:

1. The CSP informs the TA that the server $S(i)$ is suspicious and sends to it the proofs it has.
2. The TA verifies that the TL of the named server is less than the (d), and then requests the trace directly from $S(i)$.

```
Verification_Flow ( Suspicious Server (S_i) ){

Request(Forward-Trace()) ===> (S_i)

While (timestamp){

    if (Not-Received())
       {Revocation ((S_i), Punishement(level=1))}

    else{
            /*     Verification  1 */

       if (Hash(ID_(S_i)). NotEqualTo (extract-ID_Hash(M_(csp-to-si),
                                                        M_(si-to-csp))))
          {Revocation ((S_i), Punishement(level=2))}

       else{
               /*     Verification  2 */

          if (Hash(T(S_i)). NotEqualTo( extract-Trace_Hash(M_(si-to-csp))))
             {Revocation ((S_i), Punishement(level=3))}

          else{
                  /*     Verification  3 */

            Agent. re-Execute();
            if (Agent. getCurrentTrace(). NotEqualTo( T(S_i)))
               {Revocation ((S_i), Punishement(level=4))}

            else{
               Print (" The Cloud Server is Honest")
}}
}}
}}
```

Figure 6.8 – Pseudo-code describing the verification flow of a cloud server maliciousness

3. If the cloud server forwards the trace, the TA proceeds to a verification of all data involved in this trace, then it re-executes the agent again and compares the given trace with that provided by the CSP.
   — if they match, then the server is considered to be honest and the CSP is warned for dishonest attitude.
   — else, the server is provisionally revoked and punished by subtracting a percentage of its TL according to a punishment scale.
4. If the server did not present the requested trace, then it is revoked with decreasing its TL until sending the trace or proving its good intention.

Concerning the permanent revocation protocol, it becomes activated when the TL of the

suspicious server exceeds the threshold (d). It is defined in the steps bellow:

1. The CSP informs the TA that the server $S(i)$ is malicious and sends all its proofs to demonstrate that it conducts incorrect behaviors.
2. The TA verifies that the TL of the named server is less than the (d).
3. The TA verifies narrowly all information provided by the CSP, beginning by the timestamp freshness, the identities of communicating parties and the session key produced. Whether one of these data is incorrect, the TA stops the revocation and warns the CSP for invalid information. Else, verification is pursued.
4. The TA computes the hash of the given trace in the CSP proofs $Hash(T(S(i)))$ with the hash included in the trace of the server $S(i+1)$. If they match, it sends the mobile agent to be re-executed and verifies if the obtained trace agrees with the others.
   — if they agree, this means that the CSP claimed a revocation basing falsified proofs. In this case, the TA imposes a sanction to the CSP for its dishonest attitude.
   — else, the server is permanently revoked with decreasing its TL.

6.3 Performance Analysis

In this section, we evaluate the detection and prevention performances of our Cloud-IDPS to prove its reliability and effectiveness. For that purpose, a basic cloud environment was implanted using the simulation toolkit CloudSim **[Calhe10]**, which is configured on the Java IDE Eclipse. The CSP and the cloud servers are represented by virtual machines (VMs) allocated on physical heterogeneous hosts with the characteristics denoted in Table 6.2. According to this, many datacenters are implanted and mastered by our CSP, such that each datacenter contains several cloud servers hosted on VMs.

Table 6.2 – Technical Characteristics of the machines used in the evaluation

Characteristic	**Value**
OS	Windows_XP, Ubuntu, MacOs
Processor	Core i7 at 2.7 GHz
RAM	4 Go
Bandwidth	1000 Mbit/s

Indeed, the use of mobile agents needs the integration of an agent framework in each machine. We make use of JADE 4.3.3 FIPA-compliant agent platform, which is also configured on Eclipse and charged with receiving, executing and dispatching our mobile agent. This latter has the characteristics denoted in Table 6.3.

The experiments being conducted initiate five datacenters hosting increasingly 2, 4, 6, 10 and 15 cloud servers, as virtual machines (VMs). It is worth to mention that CloudSim involves two fundamental entities: *the brokers* responsible for managing the actions on VMs, and *the cloudlets* which are the tasks to be executed by the servers. Hence, the requested tasks are assembled in a list of cloudlets submitted by the broker, then assigned to the mobile agent before its mobility, during which the execution of these tasks is traced. Figure 6.9 shows an example of Java pseudo-code used in CloudSim to establish the proposed configuration. The evaluation of our IDPS for cloud focuses on three important metrics: response time, network

Table 6.3 – Technical Characteristics of the mobile agent

Characteristic	**Value**
Storage Memory	512 Mbits
Processor	GenuineIntel 800 MHz
Communication Protocol	HTTP-based-MTP
Bandwidth	100 Mbit/s

load and detection rates. This is achieved while making a comparison with the mathematical model presented by [**Braun05**], where a basic client/server architecture is adopted to ensure communication inside cloud.

```
public void processEvent(SimEvent ev) {
    int num_user = 1;
    Calendar calendar = Calendar.getInstance();
    CloudSim.init(num_user, calendar);
    Datacenter DC4 = createDatacenter("Datacenter_04", 10);
    DatacenterBroker broker = (DatacenterBroker)createBroker("DCBroker");
    broker.submitVmList(createVM(broker.getId(), 10, 1));
    broker.submitCloudletList(createCloudlet(broker.getId(), 7, 1);
    CSPAgent.Tasks= broker.getCloudletList;

    CloudSim.startSimulation();
    CSPAgent.move();
//...
}
```

Figure 6.9 – CloudSim code for processing one datacenter with 10 cloud servers and 7 cloudlets assigned to the mobile agent

6.3.1 Response Time

When the mobile agent is charged with manifold tasks that require mobility across numerous cloud servers, it is strongly needed, before taking the overall response time of the system, to calculate the overhead of security added through performing the authentication mechanism and the trace generation. Table 6.4 illustrates this overhead face to the increase of visited cloud servers. Knowing that only one migration from one server to another takes about $156ms$, it is clearly noticed that the authentication and trace generation represent a very low percentage of the overall time spent during the agent trip.

A comparison of response time between our cloud-IDPS and the work of [**Braun05**], regarding the increase of requested tasks, is shown in Figure 6.10. By virtue of this latter, we prove that adopting mobile agent technology for cloud environments is beneficial as it reduces response time of about 34% and provides a flexible and secure layout.

Table 6.4 – Time cost (in S) of the authentication and the cryptographic trace generation on increasing number of cloud servers

Nb of Servers	Authentication	Trace Generation
2	0.024	0.088
7	0.079	0.312
15	0.166	0.654
30	0.325	1.42
60	0.643	2.65

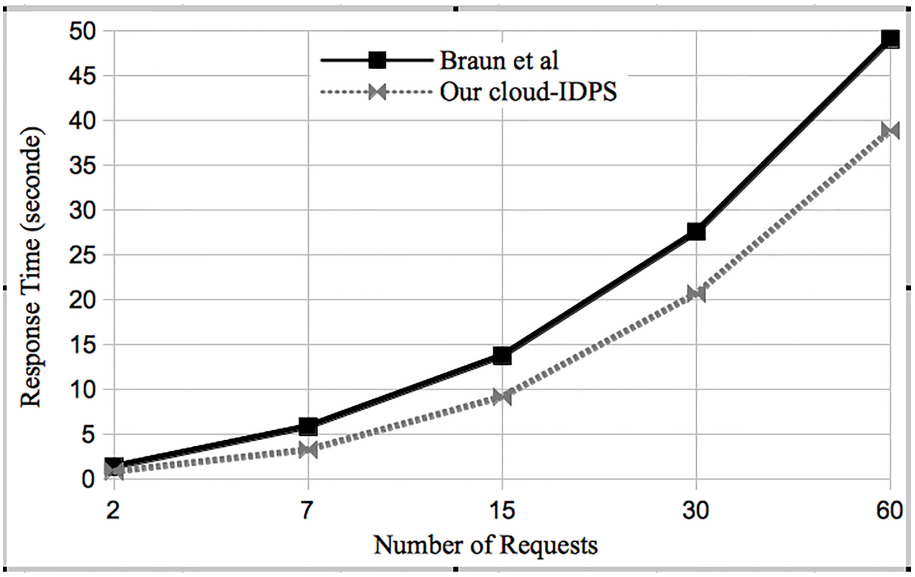

Figure 6.10 – Response Time of our cloud-IDPS compared to [**Braun05**]

6.3.2 Network Load

Our cloud-IDPS shows very challenging performance in terms of network load, compared to Braun et al [**Braun05**]. This is illustrated in Figure 6.11, where we remark that the use of mobile agent provided with security features results in a lower network load of about 30%.

It is also noted that optimum load is expressed when the number of visited cloud servers is less than six, which means that to ensure the efficiency and optimization of mobile agent tasks, it is strongly recommended that its trip do not exceed six destinations.

6.3.3 Security Performance

The efficiency of an IDPS is determined through evaluating its capacity to make correct attack detection. For that purpose, we consider two commonly used rates to quantify the detection performance:

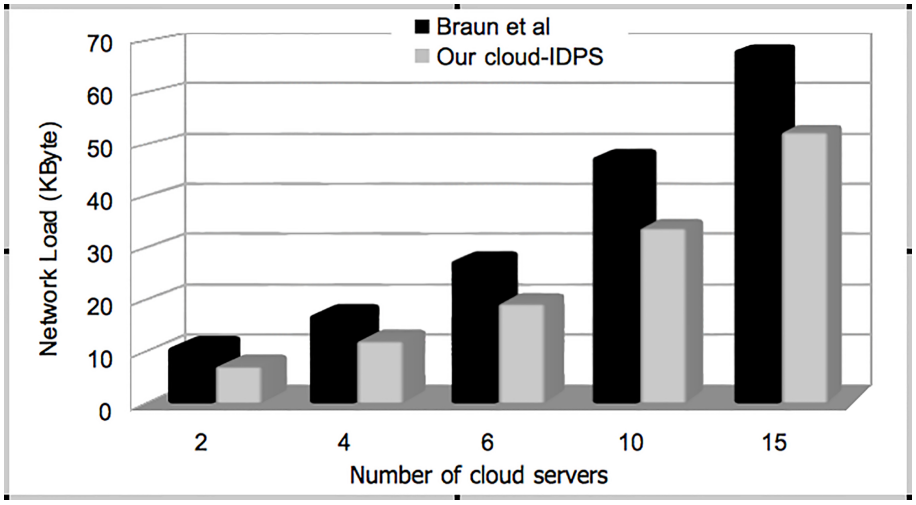

Figure 6.11 – Network Load of our cloud-IDPS compared to [**Braun05**]

$$\textit{Detection rate (DR)}: \frac{\text{Number of detected attacks}}{\text{Number of attacks}}$$

$$\textit{False Alarm rate (FAR)}: \frac{\text{Number of false alarms}}{\text{Number of alarms}}$$

For capturing and sending packets (especially malicious ones), we make use of the Java library JPCAP (Network Packet Capture Facility) [**JPCap**] with JADE [**Belli01**]. Besides, real attacks are simulated and injected in the running environments using MetaSploit tool [**Mayn11**] dedicated for penetration tests and creation of secure solutions. Examples of the used attacks are shown in Table 6.5.

Table 6.5 – Examples of simulated Attacks to evaluate detection performance

Attack	**Description**
DoS/DDoS	using a spoofed address, the attacker applies a TCP SYN flooding to burden the cloud server and make it unavailable.
Reused IP	the attacker burdens the server VM to force its disconnection and convince the centralized management component to allocate it for him.
Nmap TCP Scan	performs a scan of a remote machine to determine the available ports that can be exploited to gain shell access of the server hosting the mobile agent.

It is well acknowledged that reliable IDPS has to express a high detection rate, while keeping the false alarm rate as reduced as possible. Table 6.6 lists the results of injecting different attacks to evaluate the behavior our IDPS, compared to the system of Braun et al, while detection rates

specific for each kind of attack are shown in Figure 6.12. They clearly prove the robustness and effectiveness of our IDPS in detecting intrusions, through highly significant detection rates that are enhanced of about 15% more than Braun et al.

Table 6.6 – Comparison of detection performance between our IDPS and the system of Braun et al [**Braun05**]

Number of Cloud Servers	**2**	**4**	**6**	**10**	**15**	**30**
Number of injected attacks	1	2	3	6	10	17
Attacks detected: Braun et al	0	0	1	2	4	9
Attacks detected: our IDPS	1	2	3	6	10	17

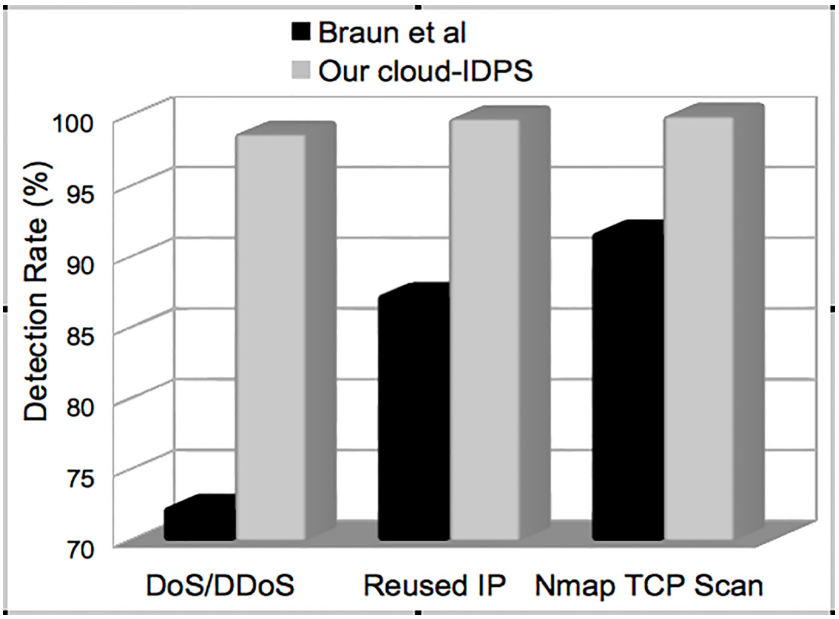

Figure 6.12 – Detection rate of our cloud-IDPS compared to [**Braun05**]

The given results are further supported by the assessment of the false alarm rate as indicated in Figure 6.13, where we mark a notably low false alarm rate of our IDPS versus a substantial rate for the system of Braun et al. Hence, our cloud-IDPS shows very promising features as security tool for cloud environment.

6.4 Conclusion

In this chapter, a new approach for intrusion detection and prevention in cloud computing is proposed. We were concerned by depicting insider threats through introducing interoperable and secure mobile agents able to trace their execution on multiple cloud servers. Provided with cryptographic mechanisms, an agent needs to save a proof containing the black statements

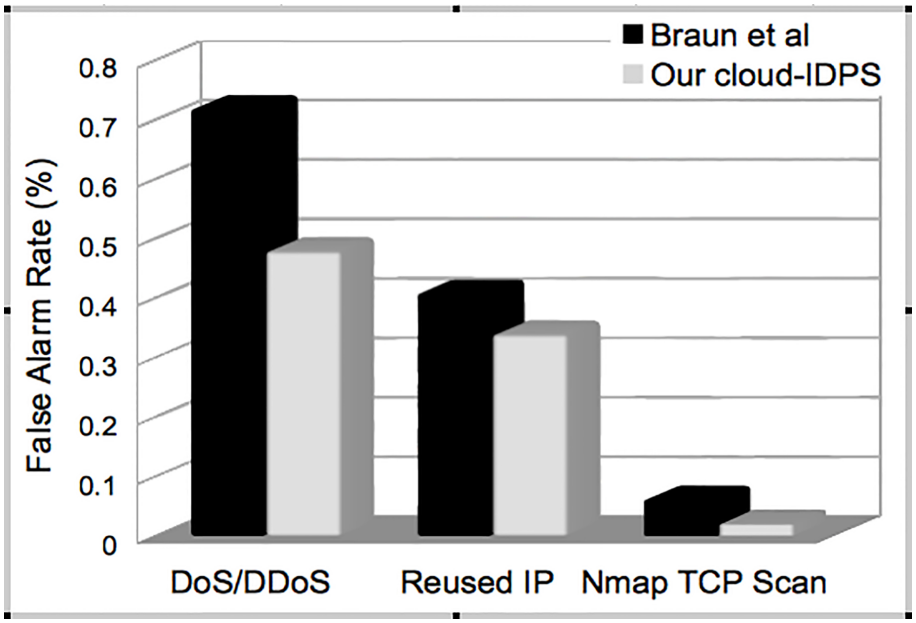

Figure 6.13 – False alarm rate of our cloud-IDPS compared to [**Braun05**]

of the tasks performed during its mobility. These traces undergo a thorough verification, in which a prevention policy is agile in case of detected maliciousness. Depending on a trust threshold, malicious servers are provisionally or permanently revoked, while decreasing their trust level for each suspicious behavior. The practical experiments we have conducted show a low response time and network load, against a high detection performance compared to the traditional client/server architecture. In future, we will be interested in extending our IDPS to cover the different layers and infrastructures of the cloud architecture, as well as introducing an access control model able to collaborate with mobile agents.

General Conclusion

The technological revolution shown in the applications based on mobile computing, especially those using the mobile agent paradigm, was essentially caused by a set of recent phenomena such as nomadic users, terminal mobility or the activity in the disconnected mode. Moreover, when these phenomena are associated to the dynamicity, autonomy and independence of the environment, then, serious security issues are raised, whose resolution represents a real challenge.

The context of this memory concerns the security of mobile agent systems, either against the attacks that can be leaded by malicious hosts or by malicious agents. Although the field of security is relatively old, this problem is far from being solved, due to the life cycle of the attacks and solutions. But it is always possible to make an attack more difficult through minimizing the risks and assuring a sufficient level of safety.

In this work, we have first defined the specific aspects of these systems which include autonomous agents able to move across the network, as well as hosting platforms allowing the execution of these agents. Besides, we have highlighted the requirements needed by every information system, particularly mobile agent system, in order to be qualified as "secure system". Then, we have conducted a thorough study on the popular security attacks, which are classified into two categories: attacks against mobile agent and attacks against platform. According to these two categories, we have discussed the counter-measures proposed in literature to investigate this problematic.

In order to provide our mobile agent system with the authentication aspect, many mechanisms were introduced such as Diffie-Hellman key exchange integrated with digital signature, ID-based key agreement protocol integrated with Schnorr signature, as well as an anonymous authentication based on elliptic curve and bilinear pairing. We were primarily concerned by counteracting Man-In-The-Middle attacks and masquerading attacks, as well as ensuring forward secrecy and key freshness. In addition, all authentication schemes proposed generate a session key, which is used lately in other processes.

Confidentiality aspect is preserved in our approaches through the use of encryption. Thus, our agent was usually encrypted before its migration, and decrypted once arrived to destination. We made use of a symmetric-key algorithm "AES", because it is robust and much faster, moreover, a session key is provided and could be used. Elliptic curve cryptography is also adopted, since it ensures efficiency, reliability, resistance to a large range of attacks, in addition to the small key lengths it supports and which are provided by a trust third party. Among the famous attacks we were concerned by neutralizing is the eavesdropping attack.

Integrity aspect is eventually ensured through signing the mobile agent including, its code and data. A variety of signatures was performed throughout the proposed approaches, according to the crypto-system involved. Thus, we make use of the digital signature (DSA), the Schnorr signature and the elliptic curve digital signature (ECDSA), with the aim to detect alteration

attacks. In order to detect any other malicious behaviors of the agent or the platform charged with its execution, we have introduced a detection intrusion mechanism based on execution tracing, such that all performed behaviors and actions are recorded and verified once the agent returns back to its owner.

Access control aspect is also considered to secure the platforms of mobile agents against the unauthorized access attacks. Thus, a discretionary access control policy is adopted along with Shamir Threshold sharing scheme, in order to simulate a key management of the access rights to the platform resources. According to this, each resource gets a set of shares as keys corresponding to its specific access rights.

It was of great interest to ensure that these mechanisms would not compromise the mobility and flexibility features of the agent. Thereby, the serialization principle, either the XML version or the binary version, was employed to guarantee a persistent context of the agent, which will facilitate its mobility and promote its visibility across the network.

With the intention to manifest the advantageous use of mobile agent security for the benefit of other technologies, we have proposed a security solution for the cloud computing. It consists in an IDPS relying on secure mobile agents moving across the cloud servers and executing tasks independently. Thus, along its trip, the mobile agent saves cryptographic traces of the services and computations it performs on each server, while a revocation protocol based on trust threshold is launched by the agent's owner once it suspects a cloud server.

Perspectives

Finally, this work provides first attempts to protect the mobile agent systems. However, many perspectives are considered in the future, such as:

— The design of a bi-directional approach, which allows in parallel, the mobile agent and the hosting platform, to evaluate the level of security of the entity being in contact. This implies the use of mechanisms which can be mutually monitored and controlled.

— The increase of the mobile agent autonomy, in a such way that it can take the initiative to react after the evaluation of the host's reliability. We think it would be of high interest to introduce the trust-based models along with learning features.

— The use of hybrid meta-heuristics to optimize the mobility of the agent in particular applications, such as the traveling salesman with one-commodity pickup and delivery.

— The use of mobile agents security in the benefit of other emerging technologies, such as: Mobile Computing, Smart Cities and JavaCards.

Bibliography

[ACL02] FIPA, A. C. L. (2002). Fipa acl message structure specification. Foundation for Intelligent Physical Agents. Available at: *http* : *//www.fipa.org/specs/fipa*00061/*SC*00061*G.html*.

[Alf05] M. Alfalayleh and L. Brankovic. "An Overview Of Security Issues And Techniques In Mobile Agents", In Communications and Multimedia Security (pp. 59-78). Springer US. (2005).

[Ann03] L. D'Anna, B. Matt, A. Reisse, T. Van Vleck, S. Schwab and P. LeBlanc. "Self-protecting mobile agents obfuscation report". Network Associates Laboratories Report. (2003).

[Auma06] J. Aumasson. "On the pseudo-random generator isaac". IACR Cryptology ePrint Archive, 2006: 438. (2006).

[Bella04] P. Bella vista, A. Corradi, C. Frederici, R. Montanari and D. Tibaldi. "Security for Mobile Agents: Issues and Challenges". Invited Chapter in the Book Handbook of Mobile Computing, I. Mahgoub, M. Ilyas(eds.), CRC Press, (2004).

[Belli01] F. Bellifemine, A. Poggi and G. Rimassa. "JADE: a FIPA2000 compliant agent development environment" .AGENTS '01: Proceedings of the fifth international conference on Autonomous agents, 216-217. ACM (2001).

[Bier02] E. Bierman, T. Pretoria and E. Cloete. "Classification of Malicious Host Threats in Mobile Agent Computing". In Proceedings of the 2002 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology (pp. 141-148). South African Institute for Computer Scientists and Information Technologists. (2002).

[Blakl11] G. R. Blakley and G. Kabatiansky. "Shamir's Threshold Scheme". In Encyclopedia of Cryptography and Security, pp. 1193-1194. Springer US, (2011).

[Blakl79] G. R. Blakley. "Safeguarding cryptographic keys". In Proceedings of the national computer conference (Vol. 48, pp. 313-317). (1979).

[Blasco10] J. Blasco, A. Orfila, A. Ribagorda. "Improving network intrusion detection by means of Domain-Aware genetic programming". In: International Conference on Availability, Reliability, and Security, pp. 327-332. (2010).

[Brah11] I. Brahmi, S. B. Yahia, P. Poncelet. "A Snort-based mobile agent for a distributed intrusion detection system". In IEEE Proceedings of the International Conference on Security and Cryptography (SECRYPT), pp. 198-207. IEEE. (2011).

[Braun05] P. Braun and W. Rossak. "Mobile agents : Basic concepts, mobility models and the tracy toolkit". Morgan Kaufmann/Elsevier and dpunkt.verlag, ISBN-10: 1558608176, USA, (2005).

[Brio01] J. P. Briot and Y. Demazeau. "Introduction aux agents: Principes et architecture des systèmes multi-agents". Collection IC2, Hermès, (2001).

[Calhe10] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. DeRose and R. Buyya. "CloudSim:A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms". Software: Practice and Experience, Wiley publishers; 41(1): 23-50. (2010).

[Can02] R. Canetti and H. Krawczyk. "Universally Composable Notions of Key Exchange and Secure Channels". Editors, Advances in Cryptology, Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), LNCS, no. 2332, Netherlands. (2002).

[Cao12] J. Cao and S. K. Das. "Mobile agents in networking and distributed computing". (Vol. 3), John Wiley & Sons. (2012).

[Che14] L. Che and X. P. Yang. "Research and Application of Mobile Agent In E-Commerce System". In Applied Mechanics and Materials, Vol. 519, pp. 458-461. (2014).

[Chen10] B. Chen and H. Cheng, "A review of the applications of agent technology in traffic and transportation systems", IEEE Transactions on Intelligent Transport Systems, vol. 2, no. 11, pp. 485-497. (2010).

[DH76] W. Diffie and M. Hellman. "New directions in cryptography". Information Theory, IEEE Transactions on, 22(6): 644-654. (1976).

[Dastj10] A. V. Dastjerdi, K. A. Bakar and S. G. H. Tabatabaei. "Distributed intrusion detection in clouds using mobile agents". In Third International Conference on Advanced Engineering Computing and Applications in Sciences, Sliema. pp. 175-180. (2010).

[Dig12] F. Dignum. "Agents for games and simulations", Autonomous Agents and Multi-Agent Systems, Springer, vol. 2, no. 24, pp. 217-220. (2012).

[East01] D. Eastlake and P. Jones. "US secure hash algorithm 1 (SHA1)". No. RFC 3174. (2001).

[Enge13] A. Enge. "Bilinear pairings on elliptic curves". arXiv preprint arXiv:1301.5520. (2013).

[Ennah14] M. Ennahbaoui, S. Elhajji, "Swot Analysis of access control models", International Journal of Security and Its Applications (IJSIA), vol.8, No.3. pp. 407-424. (2014).

[FIPA02] Foundation for Intelligent Physical Agents. "FIPA Agent Management Support for Mobility Specification", document number dc00087c. Technical report, Geneva, Switzerland, May (2002).

[Fal06] S. El-Falou. "Programmation répartie, optimisation par agent mobile". Thèse de doctorat, Université de Caen, FRANCE. (2006).

[Farm96] W. M. Farmer, J. D. Guttman, and V. Swarup. "Security for mobile agents: Authentication and state appraisal". In Proceedings of the European Symposium on Research in Computer Security (ESORICS), pages 118–130, Sep. (1996).

[Fas07] M. Fasli, "On agent technology for e-commerce: trust, security and legal issues", The Knowledge Engineering Review, vol. 1, no. 22, pp. 3-35. (2007).

[Fer99] J. Ferber. "Multi-agent systems: an introduction to distributed artificial intelligence", volume 1. Addison-Wesley Reading. (1999).

[Fiore10] D. Fiore and R. Gennaro. "Making the Diffie-Hellman protocol identity-based", Editors, Topics in Cryptology-CT-RSA, Springer Berlin Heidelberg. Proceeding of the 10th Cryptographers' Track at the RSA Conference, San Francisco, CA, USA. (2010).

[Galan11] J. Galante, O. Kharif and P. Alpeyev. "Sony Network Breach Shows Amazon Clouds Appeal for Hackers". (May 17,2011). Available: $http://www.bloomberg.com/news/2011-05-15/sony-attack-shows-amazon-s-cloud-service-lures-hackers-at-pennies-an-hour.html$

[Galla09] P. Gallagher. "Digital Signature Standard (DSS)". Federal Information Processing Standards Publication, FIPS PUB, vol. 186, no 3. (2009).

[Gav09] D. Gavalas, G. E. Tsekouras and C. Anagnostopoulos, "A mobile agent platform for distributed network and systems management", Journal of Systems and Software, vol. 2, no. 82, pp. 355- 371. (2009).

[Gilb03] H. Gilbert and H. Handschuh, "Security Analysis of SHA-256 and Sisters", Editors, Selected Areas in Cryptography Proceedings of the 10th annual workshop, SAC, August 14-15, Ottawa, Canada. (2003).

[Gom01] J. Gomuluch and M. Schroeder. "Information agents on the move". Software Focus, 2(2), 31-36. Wiley (2001).

[Green98] M. S. Greenberg, J. C. Byington, T. Holding and D. G. Harper, " Mobile Agents and Security", IEEE Communications Magazine, pp. 76-85. (1998).

[Haus00] M. Hauswirth, C. Kerer and R. Kurmanowytsch. "A secure execution framework for Java". In Proceedings of the 7th ACM conference on computer and communications security (CCS 2000), Athens, Greece, pages 43-52, (2000).

[Ho98] F. Hohl. "Time limited blackbox security: Protecting mobile agents from malicious hosts". In Mobile agents and security (pp. 92-113). Springer Berlin Heidelberg. (1998).

[Ibh11] F. T. Ibharalu, A. B. Sofoluwe and A. T. Akinwale. "A reliable protection architecture for mobile agents in open network system". International journal of computer applications, 17(7), 6-14. (2011).

[Idri14a] H. Idrissi, A. Revel and E. M. Souidi. "A New Approach based on Cryptography and XML Serialization for Mobile Agent Security". In ICAART (1), pp. 403-411. (2014).

[Idri15b] H. Idrissi, E. M. Souidi, and A. Revel. "Mobile Agent Security Using ID-Based Agreement Protocol and Binary Serialization". International Journal of Security and Its Applications, 9(5), 19-30. (2015).

[Idri15c] H. Idrissi, E. M. Souidi, and A. Revel. "Security of Mobile Agent Platforms Using Access Control and Cryptography". In Agent and Multi-Agent Systems: Technologies and Applications (pp. 27-39). Springer International Publishing. (2015).

[Idri15d] H. Idrissi, M. Ennahbaoui, E. M. Souidi and S. E. Hajji. "Mobile Agents with Cryptographic Traces For Intrusion Detection in the Cloud Computing". Procedia Computer Science, 73, 179-186. (2015).

[Ismail08] L. Ismail. " A Secure Mobile Agents Platform ". Journal of Communications, Volume 3, Issue 2, pp. 1-12. (2008).

[JPCap] Available at: $http://netresearch.ics.uci.edu/kfujii/jpcap/doc/index.html$

[Jaffar13] A. Jaffar and C. J. Martinez, "Detail Power Analysis of the SHA-3 Hashing Algorithm Candidates on Xilinx Spartan-3E", International Journal of Computer and Electrical Engineering, vol. 4, no. 5, , pp. 410-413. (2013).

[Jain00] R. Jain, F. Anjum and A. Umar. "A comparison of mobile agent and client-server paradigms for information retrieval tasks in virtual enterprises". Proceedings of the Academia/Industry Working Conference on Research Challenges, pp. 209-213, Washington, DC, USA. IEEE. (2000).

[Jan99] W. Jansen and T. Karygiannis. "Mobile agent security". Technical report National Institute of Standards and Technology, Special Publication 800-19. 1999.

[Kap08] V. Kapoor,V. S. Abraham and R. Singh."Elliptic curve cryptography". ACM Ubiquity, 9(20), 20-26. (2008).

[Kholi12] H. A. Kholidy and F. Baiardi. "CIDS: A Framework for Intrusion Detection in Cloud Systems". In 9th International Conference on Information Technology: New Generations (ITNG), Las Vegas, NV, pp.379-385. (2012).

[Lan99] D. B. Lange and M. Oshima. "Seven good reasons for mobile agents". Commun. ACM, ACM, vol. 42, 88-89. (1999).

[Lee07] J. S. Lee, C. C. Chang, P.Y. Chang and C.C. Chang. "Anonymous authentication scheme for wireless communications". International Journal of Mobile Communications, 5(5), 590-601. (2007).

[Li15] Y. Li and L. Zhu. "Data Acquisition System Research Based on Mobile Agent in Network Management". In International Conference on Computer Science and Electronic Technology (ICCSET'14). Atlantis Press. (2015).

[Liu14] J. Liu, Z. Zhang, X. Chen, K. S. Kwak, "Certificateless Remote Anonymous Authentication Schemes for Wireless Body Area Networks", IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 2, pp. 332-342, (2014).

[Lu07] T. Lu and C. Hsu. "Mobile agents for information retrieval in hybrid simulation environment". Journal of network and computer applications, 30(1), 244-264. (2007).

[M.Pich] M. Pichiliani. "Introducing XML Serialization Differences between Binary Serialization and Serialization in Java". Web site: $http://mrbool.com/differences-between-binary-serialization-and-serialization-in-java/31208$.

[MSDN] "Introduction to Code Signing", (n.d.). From Microsoft Corporation, Microsoft Developer Network (MSDN) Web site:$https://msdn.microsoft.com/en-us/library/ms537361\%28v=vs.85\%29.aspx$

[Madison03] "Signed Code", (n.d.). Retrieved December 15, 2003, from James Madison University, IT Technical Services. Web site: $http://www.jmu.edu/computing/info-security/engineering/issues/signedcode.shtml$

[Marti10] C. A. Martinez, G. I. Echeverri and A. G. C. Sanz. "Malware detection based on cloud computing integrating intrusion ontology representation," in IEEE Latin-American Conference on Communications (LATINCOM), Bogota, pp.1-6. (2010).

[Mayn11] D. Maynor. "Metasploit toolkit for penetration testing, exploit development, and vulnerability research". Elsevier.(2011).

[Mell11] P. Mell and T. Grance. "The NIST definition of cloud computing". [Recommendations of the National Institute of Standards and Technology-Special Publication 800-145]. Washington DC: NIST. (2011). Available at $http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf$.

[Men11] D. E. Menacer, H. Drias and C. Sibertin-Blanc. "The MP Architecture: towards a secure framework for mobile agents". International Journal of Agent-Oriented Software Engineering, 4(4), 390-414. (2011).

[Met11] M. Metzger and G. Polakow, "A survey on applications of agent technology in industrial process control", IEEE Transactions on Industrial Informatics, vol. 4, no. 7, pp. 570-581. (2011).

[Mills10] D. Mills, J. Martin, J. Burbank, W. Kasch. "Network time protocol version 4: Protocol and algorithms specification". No. RFC5905, June (2010).

[Milo98] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. "MASIF : The OMG Mobile Agent System Interoperability Facility". Personal and Ubiquitous Computing, 2(2): 117-129, June (1998).

[Milo99] D. S. Milojičić, W. LaForge and D. Chauhan. "Mobile objects and agents (moa)". pages 595–610. 46. (1999).

[Mitro11] D. Mitrovic, M. Ivanovic, Z. Budimac and M. Vidakovic. "An overview of agent mobility in heterogeneous environments". In Workshop Proceedings on Applications of Software Agents, page 52. (2011).

[Mousa06] A. Mousa and A. Hamad. "Evaluation of the RC4 algorithm for data encryption". IJCSA, 3(2): 44-56. (2006).

[Mub07] M. Mubarak, Z. Khan, S. Sultana, H. B. Asghar, H. F. Ahmad, H. Suguri and F. Jabeen. "A Dynamic Policy based Security Architecture for Mobile Agents". in "New Technologies, Mobility and Security", pp. 493-505, Springer Netherlands. (2007).

[Murph06] A. L. Murphy, G. P. Picco and G. C. Roman. "LIME: A coordination model and middleware supporting mobility of hosts and agents". ACM Transactions on Software Engineering and Methodology (TOSEM), 15(3), 279-328. (2006).

[Nec02] G. C. Necula. "Proof-carrying code. design and implementation". (pp. 261-288). Springer Netherlands. (2002).

[Nec98] G.C. Necula, P. Lee. " Safe, Untrusted Agents Using Proof-Carrying Code", Mobile Agents and Security, G. Vigna (Ed.), Springer-Verlag, Berlin, Allemagne, pp. 61-91. (1998).

[Og15] B. O. Oguntunde, A. O. Osofisan and G. A. Aderounmu. "Embedded Mobile Agent (EMA) for Distributed Information Retrieval". International Journal of Computer Science and Information Security, 13(3), 84. (2015).

[Over04] B. J. Overeinder, F. M. T. Brazier and D. R. A. de Groot. "Cross-platform generative agent migration". In Proceedings of the Fourth European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems, pp. 356-363. EUNITE (2004).

[Page04] J. Page, A. Zaslavsky and M. Inrawan, "Countering Security Vulnerabilities In Agent Execution Using A Self Executing Security", In Proceeding of the 3rd International Joint conference on Autonomous Agents and Multi-Agents (AAMA2004), Vol 3, pp.1486-1487, New York (USA), (2004).

[Patel10] A. Patel, Q. Qassim and C. Wills. "A survey of intrusion detection and prevention systems". Information Management & Computer Security; 18: 277-290. (2010).

[Phan05] R. W. Phan."Fixing the integrated Diffie-Hellman-DSA key exchange protocol". Communications Letters, IEEE, 9(6):570-572. (2005).

[Raz14] H. Razik and A. Hair. "Self-Adaptive Security for Mobiles Agents". International Journal of Computer Applications, 94(13).(2014).

[Reis00] H. Reiser. "Security Requirements for Management Systems using Mobile Agents". In proceeding of the Fifth IEEE Symposium on Computers and Communications: ISCC 2000, Antibes, France, pages 160-165, (2000).

[Rio98] J. Riordan and B. Schneier, "Environmental Key Generation Towards Clueless Agents", G. Vinga (Ed.), Mobile Agents and Security, Springer-Verlag, Lecture Notes in Computer Science No. 1419, (1998).

[Rober12] T. Robertazzi. "Advanced Encryption Standard (AES)". In Basics of Computer Networking, pages 73-77. Springer. (2012).

[Roth98] V. Roth. "Mutual Protection of Co-operating Agents", in Secure Internet Programming, Vitek et Jensen (Ed.), Springer-Verlag, Berlin, Allemagne, pp 26-37. (1998).

[Sam01] P. Samarati and S. Capitani de Vimercati. "Access Control: Policies, Models, and Mechanisms". In Riccardo Focardi et Roberto Gorrieri, editeurs, Foundations of Security Analysis and Design, volume 2171 of Lecture Notes in Computer Science, pages 137-196. Springer Berlin Heidelberg, (2001).

[San98] T. Sander, C. F. Tschudin, "Protecting Mobile Agents Against Malicious Host", in Mobile Agents and Security, G. Vigna (Ed.), Springer-Verlag, pp 44-60. Berlin, Allemagne (1998).

[Scarf07] K. Scarfone and P. Mell. "Guide to intrusion detection and prevention systems (idps)". NIST special publication; 800(2007), 94. (2007).

[Schn91] C. P. Schnorr. "Efficient signature generation by smart cards". Journal of Cryptology, vol. 3, no. 4, pp. 161-174. (1991).

[Scott04] D. Scott, A. Beresford and A. Mycroft, "Spatial Security Policies for Mobile Agents in a Sentient Computing Environment, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 0302-9743 (Print), pp 102-117, 19 Feb, (2004).

[Sha06] M. H. Shao and J. Zhou. "Protecting mobile-agent data collection against blocking attacks". Computer Standards & Interfaces, 28(5): 600–611. 50. (2006).

[Sham79] A. Shamir. "How to share a secret". Communications of the ACM, 22(11), 612-613. (1979).

[Shi10] A. Shibli, A. Giambruno and S. Muftic. "Security architecture and methodology for authorization of mobile agents". International Journal of Internet Technology and Secured Transactions, 2(3-4), 271-290. (2010)

[Shnei01] F. B. Schneider, G. Morrisett and R. Harper. "A Language-Based Approach to Security". In R. Wilhelm (Ed.): Informatics, 10 Years Ahead, LNCS 2000, Springer-Verlag Berlin Heidelberg, pp: 86-101, (2001).

[Smith10] D. Smith, Q. Guan and S. Fu. "An Anomaly Detection Framework for Autonomic Management of Compute Cloud Systems". 34th Annual Computer Software and Applications Conference Workshops (COMPSACW), Seoul, pp. 376-381. (2010).

[Snort] Available at: $https://s3.amazonaws.com/snort-org-site/production/document\_files/files/000/000/099/original/snort\_manual.pdf$

[Sor13] A. Soriano, E. J. Bernabeu, A. Valera and M. Vallés. "Multi-agent systems platform for mobile robots collision avoidance". In Advances on Practical Applications of Agents and Multi-Agent Systems (pp. 320-323). Springer Berlin Heidelberg. (2013).

[Sri14] S. Srivastava and G. C. Nandi. "Self-reliant mobile code: a new direction of agent security". Journal of Network and Computer Applications, 37, 62-75. Elsevier (2014).

[Tai99] H. Tai and K. Kosaka. "The Aglets project". Commun. ACM, 42, 100- 101. (1999).

[Talia11] D. Talia. "Cloud Computing and Software Agents: Towards Cloud Intelligent Services". In WOA, Vol.11, pp.2-6. (2011).

[Tan02] H. K. Tan and L. Moreau, "Extending Execution Tracing for Mobile Code Security". In K. Fischer and D. Hutter (Eds.), Proceedings of Second International Workshop on Security of Mobile Multi-Agents Systems (SEMAS'2002), pp. 51-59, Bologna, Italy. (2002).

[Tanen07] A. S. Tanenbaum, A. S. "Modern Operating Systems". Prentice Hall Press, Upper Saddle River, NJ, USA. 44. (2007).

[Tau12] C. J. Tauro, N. Ganesan, S. Mishra and A. Bhagwat, "Object Serialization: A Study of Techniques of Implementing Binary Serialization in C++, Java and. NET", International Journal of Computer Applications, vol. 6, no. 45, (2012).

[Tupak11] U. Tupakula, V. Varadharajan and N. Akku. "Intrusion Detection Techniques for Infrastructure as a Service Cloud". IEEE International Conference on Dependable, Autonomic and Secure Computing pp. 744-751. (2011)

[Vigna03] G. Vigna. "Mobile agents and security", Vol. 1419. Springer, Edition (2003).

[Vigna98] G. Vigna. "Cryptographic traces for mobile agents". In Mobile agents and security (pp. 137-153). Springer Berlin Heidelberg. (1998).

[WampServ] WampServer, http://www.wampserver.com

[Wro02] G. Wroblewski. "General Method of Program Code Obfuscation". Doctoral dissertation, PhD Dissertation, Wroclaw University of Technology, Institute of Engineering Cybernetics. (2002).

[Xiong14] H. Xiong. "Cost-effective scalable and anonymous certificateless remote authentication protocol". Information Forensics and Security, IEEE Transactions on, 9(12), 2327-2339. (2014).

[Young97] A. Young, M. Yung, "Slinding Encryption : A Cryptographic Tool for Mobile Agents", Proceedings of the 4'" International Workshop on Fast Software Encryption. Biham (Ed.), Springer-Verlag, pp 230-241. Berlin, Allemagne. (1997).

[Zha01] M. Zhang, A. Karmouch, and R. Impey. "Towards a secure agent platform based on FIPA". In Proceedings of the 3th International Workshop on Mobile Agents for Telecommunication Applications (MATA '01), volume 2164/2001 of LNCS, pages 277–289, Montreal, Canada. Springer Berlin/Heidelberg. (2001).

[Zhan06] X. Zhang, F. Parisi-Presicce and R. Sandhu. "Towards Remote Policy Enforcement for Runtime Protection of Mobile Code Using Trusted Computing". Advances in Information and Computer Security, vol. 4266, pp. 1611-3349. (2006).

[Zhao14] Z. Zhao. "An efficient anonymous authentication scheme for wireless body area networks using elliptic curve cryptosystem". Journal of Medical Systems, 38(2), 1-7. (2014).

List of Publications

1. **Hind Idrissi**, Arnaud Revel and El Mamoun Souidi (**2016**). "Security of Mobile Agent Platforms using RBAC based on Dynamic Role Assignment ". International Journal of Security and Its Applications, 10(4), 117-134.
2. **Hind Idrissi**, Mohammed Ennahbaoui, El Mamoun Souidi, Said El Hajji and Arnaud Revel (**2016**). "Secure and Flexible RBAC Scheme Using Mobile Agents". In Proceedings of the Mediterranean Conference on Information and Communication Technologies 2015 (pp. 447-455). Springer International Publishing.
3. **Hind Idrissi**, Mohammed Ennahbaoui, El Mamoun Souidi and Said El Hajji (**2015**). "Mobile Agents with Cryptographic Traces For Intrusion Detection in the Cloud Computing". Procedia Computer Science, 73, 179-186.
4. Mohammed Ennahbaoui, **Hind Idrissi** and Said El Hajji (**2015**, November). "Secure and flexible grid computing based intrusion detection system using mobile agents and cryptographic traces". In Innovations in Information Technology (IIT), 2015 11th International Conference on (pp. 314-319). IEEE.
5. **Hind Idrissi**, El Mamoun Souidi and Arnaud Revel (**2015**). "Security of Mobile Agent Platforms Using Access Control and Cryptography". In Agent and Multi-Agent Systems: Technologies and Applications (pp. 27-39). Springer International Publishing.
6. **Hind Idrissi**, El Mamoun Souidi and Arnaud Revel (**2015**). "Mobile Agent Security Using ID-Based Agreement Protocol and Binary Serialization". International Journal of Security and Its Applications, 9(5), 19-30.
7. **Hind Idrissi**, Mohammed Ennahbaoui, El Mamoun Souidi, Arnaud Revel and Said El Hajji (**2014**, April). "Access control using mobile agents". In Multimedia Computing and Systems (ICMCS), 2014 International Conference on (pp. 1216-1221). IEEE.
8. **Hind Idrissi**, Arnaud Revel and El Mamoun Souidi (**2014**). "A New Approach based on Cryptography and XML Serialization for Mobile Agent Security". In International Conference on Agents and Artificial Intelligence (ICAART) (pp. 403-411). ACM.

Université Mohammed V
Faculté des Sciences
Rabat

UNIVERSITÉ
La Rochelle

Thèse de Doctorat

Discipline: Informatique
Spécialité: Informatique et Sécurité de l'Information
Laboratoire: Mathématiques, Informatique et Applications
Période d'accréditation: 2012-2016
Responsable du Laboratoire: Pr. Said EL HAJJI
Directeurs de Thèse: Pr. Arnaud REVEL et Pr. El Mamoun SOUIDI

Titre de la Thèse: **Contributions à la Sécurité des Systèmes d'Agents Mobiles**

Hind IDRISSI

Résumé :

Récemment, l'informatique distribuée a connu une grande évolution en raison de l'utilisation du paradigme des agents mobiles, doté d'innovantes capacités, au lieu du système client-serveur où les applications sont liées à des nœuds particuliers dans les réseaux. Ayant capturé l'intérêt des chercheurs et de l'industrie, les agents mobiles sont capables de migrer de manière autonome d'un nœud à un autre à travers le réseau, en transférant de leur code et leurs données, ce qui leur permet d'effectuer efficacement des calculs, de recueillir des informations et d'accomplir des tâches.

Cependant, en dépit de ses avantages significatifs, ce paradigme souffre encore de certaines limitations qui font obstacle à son expansion, principalement dans le domaine de la sécurité. Selon les efforts actuellement déployés pour évaluer la sécurité des agents mobiles, deux catégories de menaces sont considérées. La première catégorie concerne les attaques menées sur l'agent mobile lors de son voyage à travers des hôtes ou des entités malveillantes, tandis que la seconde catégorie traite les attaques effectuées par un agent mobile illicite afin d'affecter la plate-forme d'hébergement et de consommer ses ressources. Ainsi, il est substantiellement nécessaire de concevoir une infrastructure de sécurité complète pour les systèmes d'agents mobiles, qui comprend la méthodologie, les techniques et la validation.

L'objectif de cette thèse est de proposer des approches qui fournissent cette technologie avec des fonctionnalités de sécurité, qui correspondent à sa structure globale sans compromettre ses capacités de mobilité, l'interopérabilité et l'autonomie. Notre première approche est basée sur la sérialisation XML et des primitives cryptographiques, afin d'assurer une mobilité persistante de l'agent ainsi qu'une communication sécurisée avec les plates-formes d'hébergement. Dans la seconde approche, nous avons conçu une alternative à la première approche en utilisant la sérialisation binaire et la cryptographie à base de l'identité. Notre troisième approche introduit l'aspect d'anonymat à l'agent mobile, et lui fournit un mécanisme de traçage pour détecter les intrusions le long de son voyage. La quatrième approche a été développée dans le but de restreindre l'accès aux ressources de la plate-forme de l'agent, en utilisant une politique de contrôle d'accès bien définie à base la cryptographie à seuil. A ce stade, on s'est intéressé à expérimenter l'utilité des agents mobiles avec des fonctionnalités de sécurité, dans la préservation de la sécurité des autres technologies, telles que le Cloud Computing. Ainsi, nous avons proposé une architecture innovante du Cloud, en utilisant des agents mobiles dotés de traces cryptographiques pour la détection d'intrusion et d'un protocole de révocation à base de seuil de confiance pour la prévention.

Mots clés : Sécurité des Agents Mobiles, Cryptographie, Sérialisation, Contrôle d'accès, Détection et Prévention d'Intrusion, JADE.