Toward a Security Architecture for Smart Messages: Challenges, Solutions, and Open Issues *

Gang Xu[1] †, Cristian Borcea[1], and Liviu Iftode[2]

[1] *Department of Computer Science, Rutgers University, Piscataway, NJ, 08854, USA*
[2] *Department of Computer Science, University of Maryland, College Park, MD, 20742, USA*
{*gxu, borcea*}*@cs.rutgers.edu, iftode@cs.umd.edu*

Abstract

Smart Messages (SMs) are migratory execution units used to describe distributed computations over mobile ad hoc networks of embedded systems. The main benefits provided by SMs are flexibility, scalability, and the ability to perform distributed computations over networks composed of heterogeneous, resource constrained, unattended embedded systems. A key challenge that confronts SMs, however, is how to define a security architecture that protects both the SMs and the hosts, while preserving the SM benefits.

In this paper, we present a basic SM security architecture which sets up a framework for the security related issues of SMs and provides solutions for authentication, authorization, and secure SM migration. Since this paper is the first attempt to investigate the unique security challenges posed by a system based on mobile code executed over mobile ad hoc networks, we also discuss the main issues that remain to be solved for a more comprehensive SM security architecture.

1. Introduction

Programming user-defined distributed applications for large scale, ad hoc networks of embedded systems (NES) poses a significant challenge due to the unique characteristics exhibited by these networks. We envision future NES composed of a large number of heterogeneous, resource constrained systems, which are able to communicate through wireless interfaces. These nodes can be mobile, can fail at any moment, or can even be disposable. Therefore, NES will be formed ad hoc and their resources will be unknown a priori. Sensor networks [13, 11] have represented the first step toward this vision, but we expect to encounter such networks in any aspect of our daily routine (e.g., home appliances communicating to each other, cars cooperating to adapt to traffic conditions, intelligent cameras performing object tracking over large areas).

Recently, we have proposed cooperative computing [5] as a new distributed programming model for large scale, ad hoc NES. Applications developed under this model are composed of cooperative Smart Messages (SMs). SMs are collections of code and data that migrate through the network, one hop at a time, executing at each node. Nodes in the network support SMs by providing a virtual machine and a name-based memory region, called Tag Space. The applications need to execute on target nodes named by their content, and in doing so they migrate to nodes of interest using application controlled routing executed at intermediate nodes.

SMs' design has been influenced by a variety of other research efforts, particularly mobile agents for IP-style networks [14, 9, 19]. We leverage the general idea of code migration, but we focus more on flexibility, scalability, reprogrammability, and the ability to perform distributed computing for unattended NES. Section 6 describes the similarities and differences between SMs and mobile agents in more details.

This paper addresses a key challenge that confronts SMs: how to define a security architecture, while preserving the flexibility offered by SMs and avoiding a drastic degradation of performance. Although the security for both mobile agents [10, 15] and ad hoc networks [22, 12] have been extensively studied, this is the first attempt (to the best of our knowledge) to investigate the security issues for a system based on code migration over mobile ad hoc networks. Given the complexity of the problem, our intentions are threefold: identify the unique challenges faced by such a system, design an extensible framework for SM security which provides solutions for the basic, but critical SM security requirements, and present the open issues that remain to be solved.

Similar to mobile agents, there are three main issues that

*This work is supported in part by the NSF under the ITR Grant Number ANI-0121416

†The author is also affiliated with AT&T Labs, Middletown, NJ, 07748, USA

have to be solved: (1) protecting recipient hosts from SMs, (2) protecting SMs from each other, and (3) protecting SMs from malicious hosts. These problems become more severe for SMs due to the volatile nature of their target networks. Unlike traditional mobile agents for relatively stable IP-based networks, the SMs have to overcome the lack of an infrastructure or a central authority, specific to mobile ad hoc networks, which increases significantly the difficulty of key authentication and group management.

Additionally, SMs have a number of unique features that influence the design of a security architecture. First, SMs have a unified data model (the Tag Space) which provides a single point of access to system resources. Second, no direct communication is allowed among SMs, and the only communication channel is through the shared Tag Space. Third, the SM execution is non-preemptive. And fourth, end-to-end encryption is not possible for all components of an SM since an SM has to execute at each hop in the path (i.e., it has to execute at least the routing).

The rest of this paper is organized as follows. We start by providing an SM overview in Section 2. Section 3 presents the security challenges faced by SMs. In Section 4, we describe a basic security architecture for SMs, while Section 5 discusses the open issues and future work. Related work is presented in Section 6, and the paper concludes in Section 7.

2. Smart Messages

Smart Messages (SMs) are migratory execution units consisting of dynamically assembled code and data sections, termed "bricks" and a lightweight execution state. SMs execute on nodes of interest named by content, which are discovered dynamically using application-controlled routing. The routing code is executed at each node in the path toward a node of interest [4].

Since nodes in NES are resource constrained and have limited functionality, the goal of the SM system architecture is to keep the support required from nodes in the network to the minimum, placing intelligence in SMs rather than in individual nodes. Figure 1 presents the common system support provided by nodes to SMs: a name-based memory region, called Tag Space, a Virtual Machine (VM), and an Admission Manager. The Admission Manager receives incoming SMs, decides whether or not to accept them, and stores these messages in the SM ready queue. The VM acts as a hardware abstraction layer for executing tasks generated by accepted SMs. The Tag Space offers a name-based memory, persistent across SM executions, and a uniform interface to the host OS and I/O system.

Each SM has a three-stage life cycle: (1) creation, (2) migration to a node of interest, and (3) execution upon acceptance at destination. After completing its execution at a node, an SM may terminate or may decide to migrate to other nodes of interest.

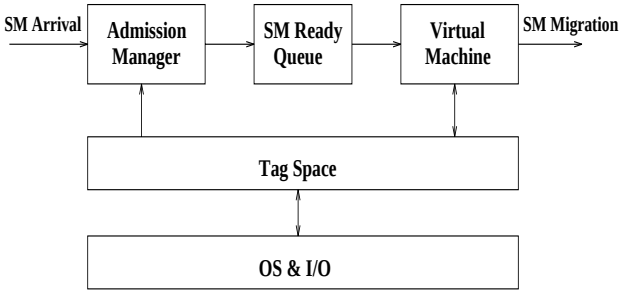

Figure 1. Node Architecture

Since SM tasks do not hold any resources at nodes (e.g., files, sockets) and there is no direct sharing among SMs, it is possible to implement a lightweight migration. The current execution control state is captured and migrated along with the code and data bricks (i.e., the only data transferred is the one incorporated explicitly by programmers in the data bricks). To reduce the cost of transferring the code, the nodes cache the code bricks.

To successfully migrate, an SM must be admitted on the remote node. The Admission Manager performs admission control at nodes in order to prevent excessive use of resources (processor cycles, memory, energy, network bandwidth). The acceptance decision is based on the estimated resource requirements presented by SMs, and the VM ensures that a task conforms to its declared requirements.

Upon admission, an SM generates a task which is scheduled for non-preemptive execution (other SMs can be accepted, but not scheduled until the current SM terminates). The execution starts from where it was left before a migration or from the beginning for new SMs.

The Tag Space is a name-based memory maintained by each node and is persistent across SM executions. Essentially, each tag consists of a triplet *(name, lifetime, data)*. The name field is similar to a file name in a file system. The lifetime specifies the duration after which the tag will expire. The data field is application defined. The Tag Space contains two types of tags: application and I/O tags. The application tags are created dynamically by SMs. The I/O tags are predefined on nodes and they provide SMs with a unique interface to the local OS and I/O system. The tags can be used for naming, routing, synchronization, data exchange, or data sharing among applications.

We have implemented an SM platform by modifying Sun's Java K Virtual Machine (KVM). KVM is a virtual machine designed for mobile devices with resource constraints and with as little as 160KB of memory. The SM applications are written in Java, and the SM API is implemented as Java libraries using native methods.

3. Security Challenges

SMs face a set of security challenges similar to those mentioned in Section 1 for mobile agents. Solving these issues becomes even more challenging for SMs since they have to cope with the highly volatile nature of their underlying networks (i.e., mobile ad hoc networks).

3.1. Protecting the host

Host security includes authenticating the owner of the incoming SM, enforcing access control based on the authentication result, and preventing resource depletion.

Identification and authentication. Authentication is necessary not only because it ensures that malicious SMs do not distribute false or useless information in the network, but also because it is a prerequisite for access control to the Tag Space. For instance, a node may only want to allow a specific group of SMs to read certain tags and keep them secret to all the other SMs. Thus, to fulfill this goal, the host has to know the ID of each SM requesting access to the Tag Space. In IP networks, authentication can be supported via a trusted server such as Certificate Authority in PKI or Key Distribution Center in Kerberos. However, having a central server in ad hoc networks is generally not feasible, and therefore the authentication becomes a more complex problem.

Authorization and resource management. A recipient host mediates the access to three types of resources: virtual machine, system resources such as I/O devices, and memory (i.e., both I/O and memory are accessed through the Tag Space). In addition to making the authorization decision, the recipient host must supervise the resource utilization in real-time to avoid the depletion of resources, which can happen in case of a Denial-of-Service attack. The protection of the Tag Space is the liability of the recipient host to cooperative SMs (i.e., the owner of a tag determines the access policy and the host enforces it).

3.2. Protecting Smart Messages against each other

SMs can interact only through shared tags. Assuming that VM is safe and complies with the SM protocols, protecting SMs from each other is boiled down to Tag Space protection and resource management for competing SMs. The former was mentioned above, and the latter requires means of coordinating resource consumption among concurrent SMs.

3.3. Protecting Smart Messages from hosts

Similar to protecting static messages, the protection of SMs needs to achieve goals such as privacy, integrity, and non-repudiation. This is especially important because of the inherent insecurity of wireless networks which are vulnerable to eavesdropping. Unlike static messages that carry constant data, SMs also carry code and dynamic data, which must be protected as well. Securing constant data is relatively straightforward using cryptographic methods such as encryption and digital signatures. Code bricks, on one hand, never change during the life cycle of an SM. Therefore, ensuring its tamper-proofness or integrity is achievable through message digests. On the other hand, it is hard if not impossible to hide any secret about the program logic from analysis attacks performed by the host where the code runs (i.e., privacy of the code bricks to the recipient host is unavailable if they need to be executed on the host). Dynamic data, such as data bricks and state information, has the same problems with the code because it must be understood by the recipient host, and differ in that they may change during the SM execution. As a result, both privacy and integrity are only possible in hop-by-hop fashion. A key issue is how to securely exchange keys for encryption and validation in ad hoc networks where no central trusted server can be relied on.

4. Basic Security Architecture

In this section, we present a basic security architecture for SMs, which does not intend to overcome all challenges and come up with a complete solution. Instead, it sets up an extensible framework and provides lightweight safeguards to support basic, but critical security requirements of cooperative computing using SMs. Specifically, this basic architecture addresses the first two challenges discussed in the previous section (i.e., we do not address the protection of SMs against malicious hosts, except for eavesdropping). Next section will cover some of the open issues and will provide an insight into possible solutions for them.

4.1. Assumptions

We assume that a key exchange mechanism is available. Secure key exchange over ad-hoc networks is an active research topic by itself [3, 21, 22], and although very challenging, it is outside the scope of this paper. A simple solution for medium-scale networks is through an off-band channel.

We further assume that each node carries a public-private key pair, and that digital signature based authentication can be implemented. Additionally, each node uses the same one-way hash function, and each code brick of an SM is identified by its hash value (i.e., the value returned by applying the hash function stored at each node on the code itself).

Finally, since our virtual machine for SM execution is an extension of Sun's Java KVM, we rely on Java language to provide both type safety (i.e., strong type checking) and memory safety (i.e., forge-proof pointers).

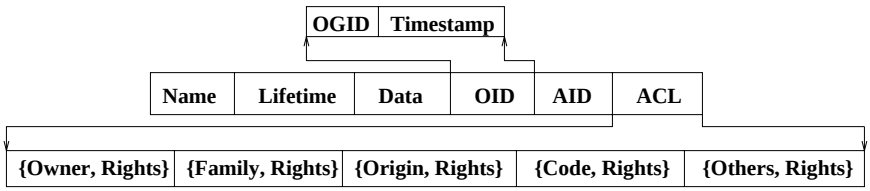

Figure 2. Tag Structure

4.2. Smart Messages Identity

The ID of an SM is a unique number, which consists of the ID of the originator node (OGID) and a timestamp. Since an SM can migrate on its own and create or spawn new SMs at intermediate nodes, its family information might be needed for access control to the Tag Space. A family of SMs is defined as all SMs generated from the same SM, called ancestor of the family (i.e., the code bricks carried by a child SM represent a subset of its parent's code bricks). Therefore, each SM carries its ancestor ID (AID). In addition, an SM carries a list of the hash values for all its code bricks, which are used for more flexible access control to the Tag Space. For instance, some routing tags may be accessible to all SMs using the same routing brick. Such an access policy can easily be verified using the brick level hashes. Also, these hashes serve as indexes, during migrations, to retrieve only the code bricks that are not cached locally. Once all this information needed for authentication and access control is generated, the sender host digitally signs it.

4.3. Authentication

Before any SM migration, the sender host authenticates the recipient host (i.e., verifies if the recipient is trusted or not). The ID, AID, and hash list of the SM are encrypted using sender's private key and decrypted by the recipient with sender's public key. The SM's code, data, and execution state might be encrypted with the recipient's public key and decrypted at destination with the private key. This solution, however, can be computationally expensive because the SM has to be encrypted/decrypted at each node in the path toward a node of interest (i.e., SMs execute their routing at each node in the path). Therefore, Section 5 discusses an approach that allows an SM to pass through the intermediate nodes in an encrypted form (except for its routing and execution state) and be decrypted only at the node of interest. Upon receiving an incoming SM, the recipient host checks if the digital signature is valid and made by a trusted party. Then, the integrity of the SM is verified using the list of hashes. If any code brick is detected to be modified, the SM is considered being tampered and thereby not trusted.

4.4. Access Control

A unique characteristic of SMs is that no direct access is allowed to system resources (i.e., the SMs access both their data and system resources through the Tag Space). The advantage of this design is that the Tag Space is a single point of access control, which can be implemented and enforced uniformly. Compared to other systems [10], it greatly simplifies the control mechanisms. As described in Section 2, there are two types of tags: I/O tags and application tags. It is the host's responsibility to define policies to protect the I/O tags and the implementation is straightforward. The application tags are the only mean of communication and coordination among cooperative SMs. The SM creating the tag, called tag owner, determines the access control policy and delegates the host to enforce this policy on its behalf. Protecting the application tags ensures that SM executions do not interfere with each other, and therefore provides a secure channel for cooperation.

Besides its name, lifetime, and data, a tag incorporates also the ID of its owner (OID), the ID of its owner's ancestor (AID), and the ACL (access control list), as depicted in Figure 2. Upon creating a tag, the VM sets the OID field to the ID of the SM that created the tag. Then, the SM establishes the ACL, which is a matrix of subjects and their access permissions. Access permissions for tags are similar to those for Unix files including read(r) and write(w) (i.e., to execute a system call, a read or a write is performed on an I/O tag). The ACL contains five protection domains: *Owner*, *Family*, *Origin*, *Code*, and *Others*. The *Owner* and *Others* protection domains define the access permissions for the owner of the tag and for any SM, respectively. The group concept, defined as an arbitrary relation over SMs, supports more flexible cooperation, but also requires high overhead of managing the group membership on-the-fly. Currently, our architecture does not support dynamic cooperation among totally independent SMs (this issue is discussed in more details in Section 5). Instead, we define three protections domains that allow cooperation among well-defined groups of SMs (i.e., *Family*, *Origin*, *Code*). In the following, we present three scenarios that illustrate these protection domains.

Family cooperation. In Figure 3, all cooperative SMs originate from a common SM ancestor. For instance, *SM1*

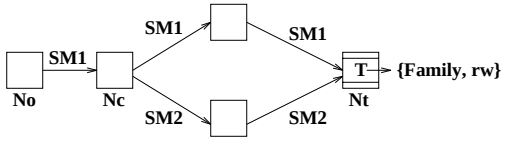

Figure 3. Family Cooperation

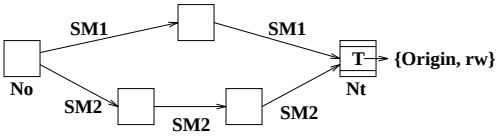

Figure 4. Single Originator Cooperation

is created on *No*, migrates to *Nc* and creates a child, *SM2*, to help it discover a route. Once a route is discovered, *SM1* needs permissions to access the tags created by *SM2*. Therefore, *SM2* sets the ACL to $\{Family, rw\}$ (i.e., the AID of *T* is the same with the AID of *SM1*).

Single originator cooperation. Figure 4 shows the scenario when the group of cooperative SMs originate from a common node. *SM1* and *SM2* are created on node *No* and migrate to target node *Nt*, via different paths. *SM1* arrives at *Nt* before *SM2* and creates a tag *T*. It also sets the ACL as $\{Origin, rw\}$ such that *SM2* will be able to access *T* (i.e., the unique IDs of *SM1* and *SM2* contain the same OGID). This scenario is very likely to be encountered since many nodes are small devices, such as PDAs or cell phones, owned by a single user.

Code-based cooperation. In addition to the simple groups described before, the SM group cooperation can be coordinated more flexibly based on code bricks. To ensure cooperation among SMs that are aware of the code used for data sharing or data exchange, each tag has a list of associated hash values for certain code bricks. These hash values define the members of the *Code* group (they may or may not belong to the owner of the tag). By definition, an SM is a member of the *Code* group if the hash value of its currently executing code brick belongs to this list. For instance, SMs using the same routing brick can add the hash value corresponding to this brick to the tag's list of hash values in order to facilitate route sharing among them. Figure 5 presents such an example. *SM1* creates a tag *T* and sets the ACL to $\{Code=(Cr), rw\}$ to grant access to all the other SMs using the *Cr* routing brick. Hence, *SM2* has the permissions to use *T*. Another example of code-based cooperation is an SM producer-consumer application (independent SMs, created on different nodes, but aware of the code bricks that access the shared data) that attaches the hash values of the code bricks used to share data to certain tags of interest.

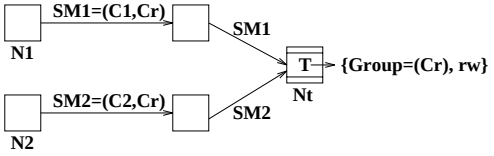

Figure 5. Code-based Cooperation

4.5. Authorization

Each time an SM tries to execute an operation on a tag, the VM performs the authorization process. Based on the credentials presented at authentication and the currently executing code brick, the SM is associated with at least one protection domain. The request is granted if the SM has the necessary permission to access the tag in any of the protection domains it has been associated with.

4.6. Resource Consumption Control

To address the challenge posed by resource depletion at nodes and to ensure a simple resource management mechanism among competing SMs, we implement a contract-based resource control mechanism. Each SM carries its resource requirement estimates in a resource table, which works as a resource contract between the recipient host and the message. As explained in Section 2, the Admission Manager decides whether or not to accept the SM, and the VM enforces the contract. Since SM execution is non-preemptive, once an SM is admitted, the recipient node guarantees that enough resources are available for its execution. Additionally, the non-preemptive execution avoids the potential problem of multiple SMs competing for resources.

5. Open Issues and Future Work

Security of both mobile agents and ad-hoc networks is known to be difficult. The combination of the two makes the solutions more elusive. Therefore, the SM security architecture is far from complete and leaves several open issues.

A first issue is how to achieve key exchange and validation without a central authority. It is clear that the tree-like trust which is the foundation of public key infrastructure [2] is not suitable for mobile ad-hoc environments. Instead, the web of trust as first described in PGP [8] seems more promising. Furthermore, to counter the potential vulnerability of having one malicious certificate authority subvert the whole system, fault-tolerance [26] and voting protocols [18] can be applied. A main drawback of these methods is the overhead in terms of message costs. We plan to investigate solutions that include achieving asymmetric proper-

ties using symmetric cryptography [21], and combining ID-based and threshold cryptography as suggested in [16].

Given the wireless nature of the underlying network, an SM has to be protected against eavesdropping. Since most of the time only the SM routing is executed (i.e., the rest of the code executes only on nodes of interest), there is no need to pay the cost of using hop-by-hop encryption/decryption for all code and data bricks. A more efficient solution would be to allow the routing brick to encrypt all the other code and data bricks using a routing-specific algorithm and a key generated based on the SM ID. Upon arrival at a node of interest, the routing decrypts the SM using the same key. The routing code and data, as well as the execution state, are still encrypted with the next hop public key and decrypted with the private key at destination.

As we mentioned in Section 3, there is no way for a host to predict on which hosts the SMs will be executed since SMs are autonomous on their way. The direct consequence is that it is hard to prevent an SM from being tampered by a malicious host by simply encrypting the message. Hardware solutions [1, 20] represent an option. The concern with this solution is the extra cost of the devices. Complete software solutions are not known yet, but code confusion and encryption techniques are investigated [7, 24] to achieve agent security.

A significant issue that we plan to address is how to support arbitrary dynamic cooperation among totally independent SMs (i.e., a more flexible access control to the Tag Space is needed). The problem can be reduced to group membership management in ad-hoc networks. The current available protocols rely on either certificate [17] or distances [23], but they are constrained by the availability of a certificate authority and provide only limited flexibility for group description.

Finally, the network as a whole has to be protected against malicious SMs that consume too many resources (e.g., an SM that loops forever through the network, but respects its resource contract at each node). We are currently investigating a market-based approach [10] where each SM acquires off-line or on-demand a number of tokens which are used ”to pay” for the resources consumed in the network.

6. Related Work

SMs are influenced by the design of mobile agents for IP-style networks [14, 9, 19]. A mobile agent names nodes by fixed addresses, knows the network configuration a priori, and relies on the underlying network to assure the transport between nodes. Unlike mobile agents, SMs address nodes by content, discover the network configuration dynamically, and are responsible for their own routing. Furthermore, since the resources possessed by nodes are limited, SMs define a system architecture that requires minimal system support at nodes. SMs apply the general idea of code migration, but focus more on flexibility, scalability, re-programmability, and ability to perform distributed computations over ad hoc networks of resource constrained embedded systems.

SMs and mobile agents share several security issues. Among them is host protection, for which mobile agents have proposed solutions based on cryptographic authentication of the agent’s owner. Examples of such systems are Telescript [25], IBM Itinerant Agents [6], Ajanta [15], and D’Agents [10]. The existing solutions control the access to resources on the recipient host by using capability lists or access control lists (ACL). The former is carried by mobile agents themselves and checked by the recipient host. Telescript implements this solution. The latter is a preconfigured policy residing on, and enforced by the recipient host. D’Agents currently supports access control lists, but will eventually support both.

Resource management is an issue which was overlooked and begins to catch the attention of a growing number of agent systems. For instance, D’Agents defines six resource managers to monitor and control the usage of consumables such as CPU time, wall-clock time and number of child agents, file systems, libraries, programs, network, screen, and each agent is limited to a finite consumption of these resources.

Protecting an agent from malicious hosts is difficult. Most agent systems walk around this problem by assuming that agents run in a trusted environment such as an organizational network. Ajanta adopts a detective strategy by recording and checking audit trails. In the meantime, a number of partial solutions are proposed such as time-limited blackbox [7] and encrypted functions [24], but few of them have been incorporated into real agent systems.

All these agent systems rely on infrastructure support and do not have the special problems of SMs, such as key authentication and group management, originating from the ad-hoc nature of their underlying network. For secure key exchange, a variety of solutions are proposed by research done in ad hoc networks. For instance, TESLA [21], a broadcast authentication protocol used in group broadcasting [22] and routing in ad-hoc networks [12], achieves asymmetric properties using symmetric MAC functions on a loosely synchronized network in order to reduce the communication and computation overhead. In [3], the authors present a password-based multi-party key agreement mechanism which extends the idea of Diffie-Hellman two-party key exchange by ensuring that all group members who share a password will finally reach an agreement on a shared session key. An example of group management in which the group leader or its delegates hold a group public-private key pair and issue a membership certificate, which binds a mem-

ber's public key to its identity, is described in [17].

7. Conclusions

In this paper, we have presented a basic security architecture for Smart Messages (SMs) that provides solutions for authentication, authorization, and secure SM migration, while preserving the benefits of SMs such as flexibility, scalability, and ability to perform distributed computations over large scale ad hoc networks of embedded systems. As an initial effort to address security in a system based on mobile code executed over mobile ad-hoc networks, this paper has described the basic security safeguards to ensure secure cooperative computing using SMs and identified the main issues that have to be solved for a more comprehensive security architecture.

References

[1] http://www.cl.cam.ac.uk/ rja14/tcpa-faq.html.

[2] C. Adams, S. Lloyd, and S. Kent. *Understanding the Public-Key Infrastructure: Concepts, Standards and Deployment Considerations*. New Riders Publishing, 1999.

[3] N. Asokan and P. Ginzboorg. Key Agreement in Ad-hoc Networks. In *Nordsec'99 Workshop*, 1999.

[4] C. Borcea, C. Intanagonwiwat, A. Saxena, and L. Iftode. Self-Routing in Pervasive Computing Environments using Smart Messages. In *Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications (PerCom)*, March 2003.

[5] C. Borcea, D. Iyer, P. Kang, A. Saxena, and L. Iftode. Cooperative Computing for Distributed Embedded Systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, pages 227–236, July 2002.

[6] D. Chess, B. Grosof, and C. Harrison. Itinerant Agents for Mobile Computing. In *IBM Tech Report RC 20010*, 1995.

[7] F.Hohl. Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts. In G. Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 92–113. Springer-Verlag, 1998.

[8] S. Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly & Associates, 1994.

[9] R. S. Gray, G. Cybenko, D. Kotz, and D. Rus. Mobile agents: Motivations and State of the Art. In J. Bradshaw, editor, *Handbook of Agent Technology*. AAAI/MIT Press, 2001.

[10] R. S. Gray, D. Kotz, G. Cybenko, and D. Rus. D'Agents: Security in a multiple-language, mobile-agent system. In G. Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 154–187. Springer-Verlag, 1998.

[11] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System Architecture Directions for Networked Sensors. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 93–104, November 2000.

[12] Y.-C. Hu, A. Perrig, and D. B. Johnson. Ariadne: a secure on-demand routing protocol for ad hoc networks. In *Proceedings of the 8th annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 12–23, 2002.

[13] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Proceedings of the Sixth annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 56–67, August 2000.

[14] N. Karnik and A. Tripathi. Agent Server Architecture for the Ajanta Mobile-Agent System. In *Proceedings of the 1998 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, July 1998.

[15] N. M. Karnik and A. R. Tripathi. Security in the Ajanta Mobile Agent System. *Software - Practice and Experience*, 2000.

[16] A. Khalili, J. Katz, and W. Arbaugh. Toward Secure Key Distribution in Truly Ad-Hoc Networks. In *IEEE Workshop on Security and Assurance in Ad-Hoc Networks*, 2003.

[17] S. Maki, T. Aura, and M. Hietalahti. Robust Membership Management for Ad-hoc Groups. In *Proc. 5th Nordic Workshop on Secure IT Systems (NORDSEC 2000)*, 2000.

[18] D. Malkhi and M. Reiter. Byzantine Quorum Systems. In *Distributed Computing*, volume 11(4), pages 203–213. 1998.

[19] D. Milojicic, W. LaForge, and D. Chauhan. Mobile objects and agents. In *USENIX Conference on Object-oriented Technologies and Systems*, pages 1–14, 1998.

[20] E. Palmer. An Introduction to Citadel - A Secure Cypto Coprocessor for Workstations. In *IFIP SEC'94*, 1994.

[21] A. Perrig, R. Canetti, D.Tygar, and D. Song. The TESLA Broadcast Authentication Protocol. In *RSA Cryptobytes*, 2002.

[22] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security Protocols for Sensor Netowrks. In *Proceedings of the 7th annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 189–199, 2001.

[23] G. Roman, Q. Huang, and A. Hazemi. Consistent Group Membership in Ad Hoc Networks. In *Proceedings of 23rd International Conference on Software Engineering(ICSE'01)*, 2001.

[24] T. Sander and C. Tschudin. Protecting Mobile Agents against Malicious Hosts. In G. Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 44–60. Springer-Verlag, 1998.

[25] J. E. White. Telescript technology: An Introduction to the Language. In *General Magic White Paper*, 1995.

[26] L. Zhou, F. Schneider, and R. van Renesse. COCA: A secure distributed on-line certification authority. Technical Report TR2000-1828, Dept. of Computer Science, Cornell University, 2000.