Mobile Code Security

Sergio Loureiro, Refik Molva, Yves Roudier
{loureiro | molva | roudier}@eurecom.fr

Institut Eurécom
2229 Route des Crêtes
Sophia-Antipolis 06560 Valbonne - France

Abstract: this article presents two aspects of mobile code security, namely the protection of hosts receiving a malicious mobile code and the protection of a mobile code within a malicious host.

Introduction

The mobile code paradigm encompasses programs that can be executed on one or several hosts other than the one that they originate from. Mobility of such programs implies some built-in capability for each piece of code to travel smoothly from one host to another. A mobile code is associated with at least two parties: its producer and its consumer – the consumer being the host that runs the code.

Mobile code systems range from simple applets to intelligent software agents. These systems offer several advantages over the more traditional distributed computing approaches: flexibility in software design beyond the well established object oriented paradigm and bandwidth optimization, just to name two of them.

As usual, increased flexibility comes with a cost that is increased vulnerability in the face of malicious intrusion scenarios akin to Internet. Possible vulnerabilities with mobile code fall in one of two categories: attacks performed by a mobile program against the remote host on which the program is executed as with malicious applets or ActiveX programs, and the less classical category of attacks due to the subversion of the mobile code and its data by the remote execution environment.

This article provides an overview of security solutions for both types of vulnerability from Java security to function hiding through proof carrying code. In a first part, we discuss mobile code models and the security threats they pose. A second part goes deeper into the details of host protection schemes, while the third part presents open issues concerning the difficult problem of mobile code protection.

Mobile Code Security Threats

A first security threat appears if the mobile code, generated by a malicious outsider, attacks the environment where it is executed. This problem bears some similarity with trojan horses, but mobile code brings up other issues: mobile code aims at transparency, automation, and a wider scale of execution; moreover, vulnerabilities of the environment are not the only targets, [DFW96] shows some examples of attacks based on flawed semantics of the underlying language.

In addition to this relatively classical scenario, mobile code introduces a completely new problem: the host itself might be the malicious party trying to subvert the mobile.

When protecting a host from potentially malicious code, code mobility imposes the following security features:

- host and mobile code bear separate identities, therefore the mobile code's origin must be authenticated;
- mobile code is exposed through the network, hence the host must verify the integrity of the mobile code it just received;
- the host does not generate the mobile code, but another party does: consequently, the actions it performs must be limited through access control and/or checked through semantic verification.

When protecting a mobile code from a potentially malicious host, code mobility implies that the program will be run under total control of the host. This means the following threats:

- spoofing through impersonation of code owner
- theft and secrecy violation through unauthorized disclosure
- integrity violation through subversion of code semantics

To prevent all three cases, data segments as well as code semantics must be protected.

Protection of a host from a mobile code

The subject of host protection has already been thoroughly studied in recent years. The first answer in this area was to simply limit the functionality of the execution environment in order to limit the vulnerabilities.

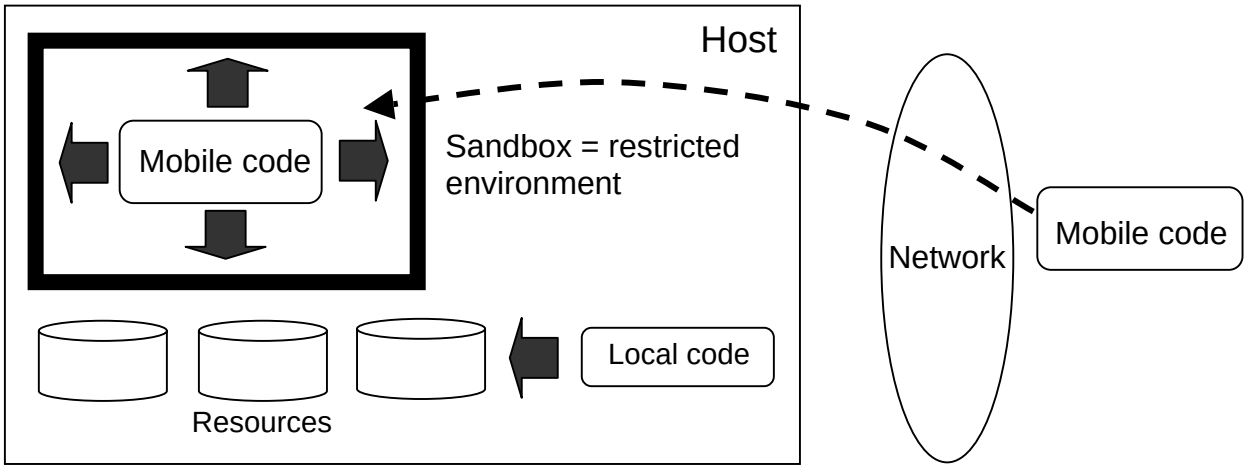
Techniques for protection of hosts now evolve along two directions:

- a mobile code infrastructure that is gradually enhanced with authentication, data integrity and access control mechanisms.
- verification of mobile code semantics.

The following sections detail both aspects.

Sandboxing

Sandboxing consists in running a mobile code in a restricted environment called the "sandbox". An otherwise untrusted mobile code can be executed without worrying in the sandbox (as shown in the figure).



A sandbox can be characterized by two different mechanisms:

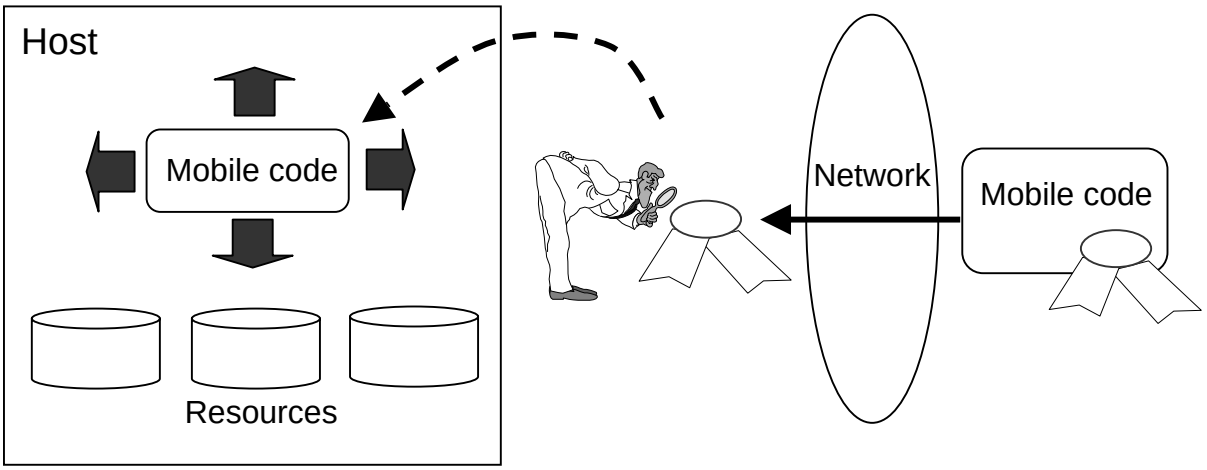
- it confines code, either through type checking, language properties, or the use of protection domains to prevent the subversion of trusted code and,
- it enforces a fixed policy for the execution of code.

This approach is especially illustrated by the early Java JDK 1.0 [GJS96], where it was used in order to enable applets available anywhere on the Internet to run within a browser.

The major drawback of sandboxing is due to the fact that applications running in such a restrictive environment are themselves seldom useful.

Code Signing

Code signing is the process by which a code is digitally signed by the code producer in order to assure strong authentication and integrity of the code to the code consumer.



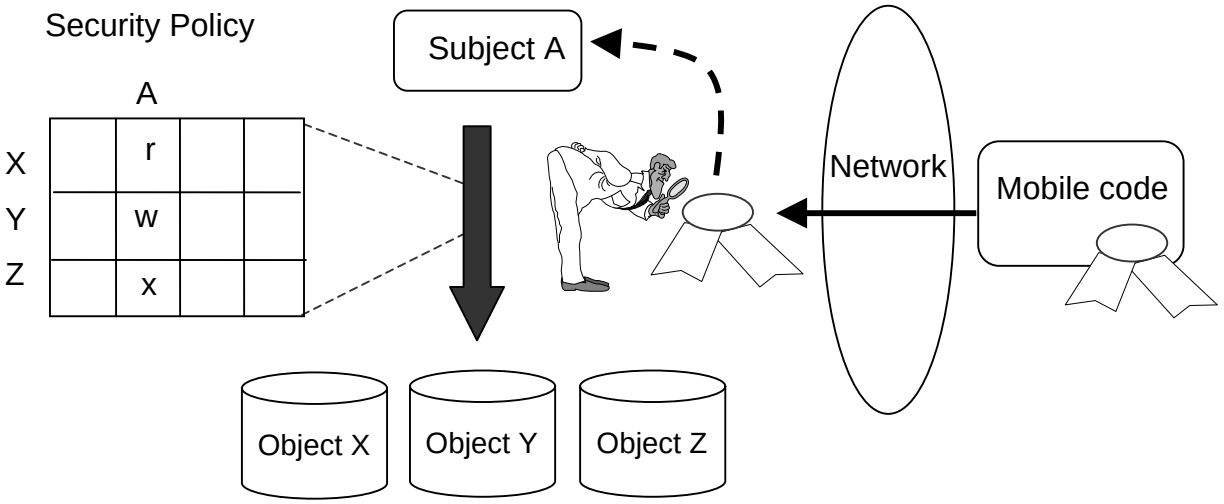
This model was first introduced by Microsoft within the ActiveX framework [Mic]. Java JDK 1.1 also follows the code signing model, with so-called *signed applets*. Upon receipt of an applet with a valid signature, the code consumer's Java virtual machine executes the applet like a trusted piece of code, authorizing it to access all features available in Java. An applet without a proper signature is run in a sandbox as in the previous version of the JDK.

There is more to securing a host from a malicious mobile code than just making sure that this program has been correctly signed by someone on the Internet. The latter simply does not imply that the code signer must be trusted unrestrictedly. Moreover, the presumption underlying code signing that users can decide whether to run a program based on a signature has questionable validity.
A related drawback of code signing lies in the rudimentary form of access control provided: a signed code is either granted full access to the resources of the code consumer, or not executed at all. This choice is left to the end-user who, even without administrator privileges, can put the entire host security at risk.

Access Control

The only way to limit this kind of security threat is to enable more complex access control schemes, in order to limit the impact of an attack. This can be seen as the refinement of a monolithic sandbox policy into smaller, application-specific policies.

The identity of the signer of a code, as described in the previous section, also helps to further refine the execution policy definition, provided that it can be established by some mean, like a public key infrastructure for instance.



Java JDK 1.2 security model [Gon98] follows this scheme and thus permits the definition of finer-grained security policies more suited to executing an untrusted mobile code.
Even more recently, Sun released the Java Authentication and Authorization Services or JAAS which aims at integrating the identity of the user running the mobile code in the definition of the access control.
Compared with sandboxing and code signing, the access control model has the best of both worlds: the actions performed by a mobile code can be restricted to some resources while it permits at the same time to write and run really useful software. Yet, the enforcement of the access scheme has a cost, since it is performed dynamically, at runtime.

Program Checking

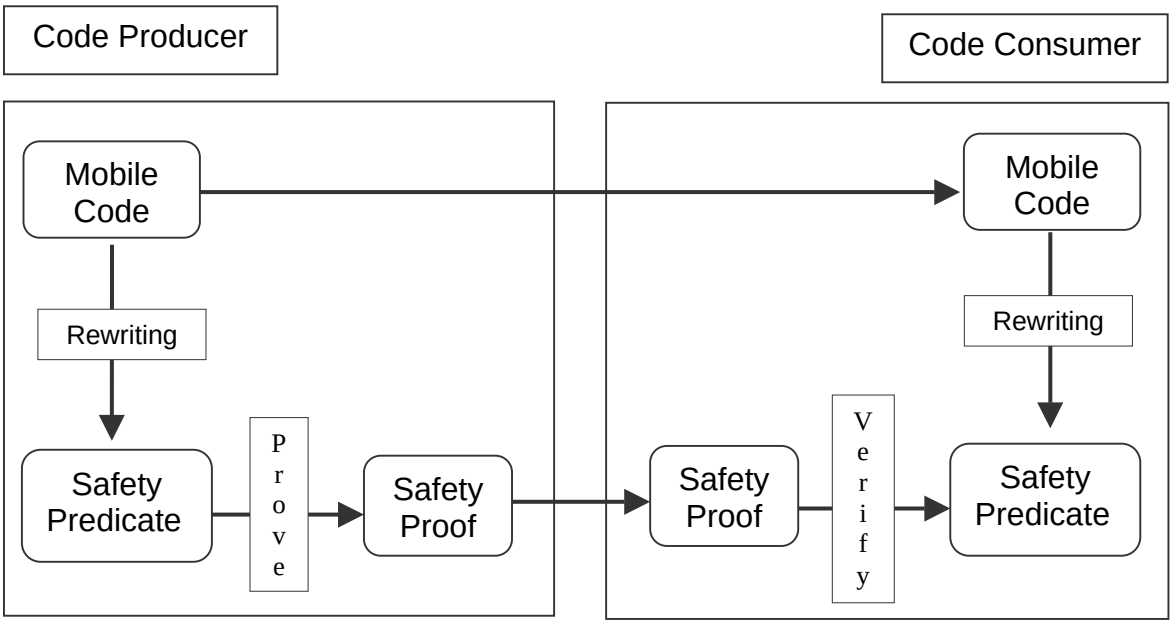
Checking a mobile code means to perform a verification on the code structure or on the code behavior as it is run and modifying in consequence the status of the code, for instance from trusted to untrusted. Checking also involves the notion of verification against a given security policy.

Sandboxes already exercise some form or another of rudimentary program check, either statically, for instance to ensure that operands of an instruction are of the correct type, or dynamically, for example to locate any access to a protected resource.

A newer approach to host protection is to statically type-check the mobile code; the program is then run without any expensive runtime checks. This approach is for instance taken in the Proof-Carrying Code scheme, and even to some extent in Java's virtual machine (for safety checks).

The Proof-Carrying Code scheme or PCC [Nec97] [NL98] is a good instance of the static program checking model. In this scheme, a predefined security policy is defined in terms of a logic. The host first asks to be sent a proof that the code respects the policy before he actually

agrees to run it. The code producer then sends the program and an accompanying proof, using a set of sound axioms and rewriting rules, as defined by the chosen logic, and shared by the code producer and the code consumer. After receiving the code, the host can then check the program with the guidance of the proof (see figure). This can be seen as a form of type checking of the program, since the proof is directly derived from it.



The most interesting part of PCC is that type-checking the proof is relatively obvious compared to proving the program and does not impose much computational burden on the code consumer. Moreover, since the proof is not the program, PCC already permits to express complex safety properties; it seems promising for security properties as well. However, automating the proof generation is still a non-trivial problem, and to date, proofs generally have to be generated by hand.

Protection of a mobile code from a malicious host

The problem of protection from a malicious host has been studied only recently, and is intrinsically more difficult because the environment gets a total control over the mobile code (otherwise, host protection would not be possible!).

The approaches used for that purpose can be classified along two criteria, 1) data versus code protection, and 2) integrity- or confidentiality-based.
Data protection is essentially interested in protecting mobile agents roaming in the Internet: an agent might encounter a malicious host occasionally who will try to brainwash the agent, for instance.
On the other hand, code protection issues address a more systematic form of maliciousness in which the environment where the mobile code runs cannot be trusted.

Data Protection

Mobile or roaming agents are a form of mobile code especially publicized in electronic commerce, but usual data protection techniques are not well adapted for protecting their data. Mobile agents are used to perform a function on behalf of the user, like for instance collect data about a product from several merchants or perform a transaction. A good example might be the so-called “comparison shopping”: for instance, an agent looks for the lowest priced book among several retailers. In that kind of scenario, it is necessary to know whether the data collected have been changed or not, if the itinerary fixed for the chain of operations has been followed. In an auction scenario, it might also be necessary to prevent a host from exploiting the offers made by the previous bidders.

The integrity of the data collected by a mobile agent might be protected using a cryptographic technique. A first possible technique is simply to digitally sign the result; the problem with this approach is that the size of the agent grows linearly as it gathers results.
A more efficient approach in terms of computational complexity and space is to use hash chaining to achieve “forward integrity”. Integrity against deletion or modification of parts of the collected data in different hosts is achieved until the first malicious host. A partial *result authentication* code (PRAC) [BY97] [KAG98], a small value, ensures the integrity of all the offers. [LMP99] extends this solution with a cryptographic technique that allows hosts to update their previous data without increasing the space requirements. These techniques have to be enhanced with digital signatures if non-repudiation is required.

On its itinerary from host to host, the agent carries an increasing amount of data that might be exploited by a malicious host. Their confidentiality can be achieved at each step of the agent itinerary by a simple enciphering on the current host, before the agent moves elsewhere. With RSA-based encryption, the security will be ensured but the data carried by the agent can be very small: the size of the data padding will then be excessive. *Sliding encryption* [YY97] aims at retaining an equivalent security, using a large key, while at the same time, taking into account the limited storage of an agent. Sliding encryption is aimed at conserving space rather than time, which might be of importance for agents that collect small amounts of data on many different hosts.

Integrity of Computation

A mobile code might be rendered “*fault tolerant*”, by performing the work to be done several times. This applies in the case of mobile agents: the same mobile agent code can be distributed to replicated hosts, and the result is discovered through a vote [MvRSS96]. Alternately, and more convincingly, it is also feasible to replicate agents and slightly alter their behavior or itinerary in order to spot a malicious host [Yee97].

A mobile code might also be protected using a cryptography-based *integrity proof*, or rather a hint of the integrity of its computation. This proof intends to guarantee that the computation was done according to the instructions of the mobile code.
The trace of the program computation is used as a form of proof, showing how the result was obtained. In order to make the proof checking process faster, this proof is transformed into a *holographic proof* [Yee97]. A holographic proof system is a system for writing and checking proofs in which one can probabilistically check the validity of a proof by examining only a few of its bits. Moreover, this checking can be performed in polylogarithmic time.

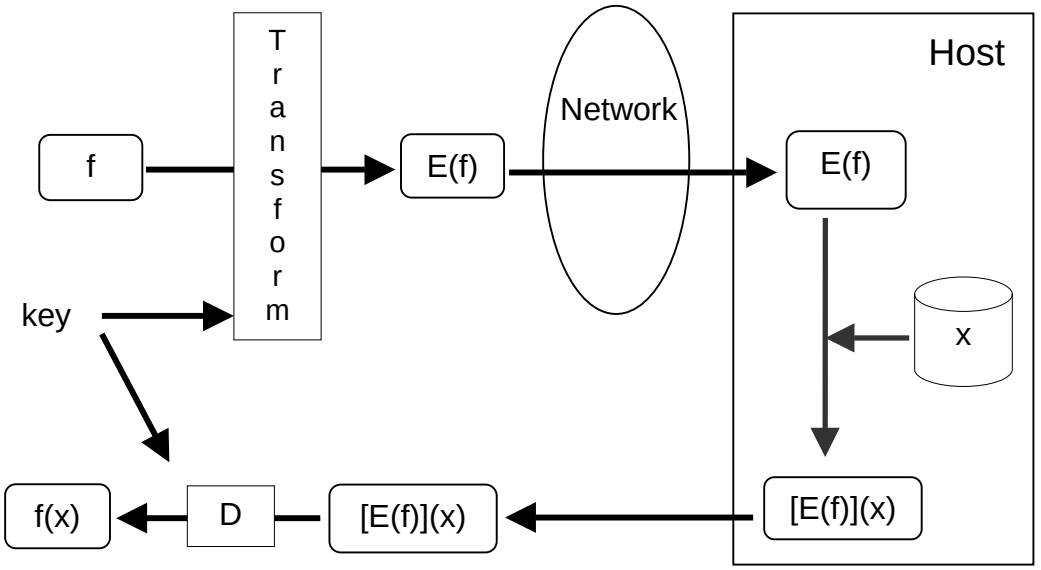
The transformation of the proof into a holographic proof is computationally intensive but it is the host where the agent is executing, and not the verifier, which performs the computation. Another

problem arises: the validity of the holographic proof is only preserved in so far as the bits of the proof queried by the verifier remain unknown to the host that performed the computation. This could be ensured if the holographic proof could be sent to the verifier host but a holographic proof is even bigger than the proof it was obtained from. This problem is solved by using a cryptographic private information retrieval technique [BMW98]: with this technique, the holographic proof remains on the host which computed it and can be queried without the host knowledge of the queried bits. The result of these hidden queries and the computation result is finally smaller than the original proof itself.

Privacy of Computation

A mobile code might be written in order to rely on *tamper-proof hardware* [WSB98], for instance a smartcard, as a secure kernel for executing secret functions and keeping secret data; in that case, the achieved services are integrity and partial confidentiality of the program.

A mobile code might finally be protected algorithmically, by hiding the computed function; this results in a standalone code, without additional hardware, achieving confidentiality of the computation. This technique is known as *function hiding.*



In the function hiding scheme (see figure), a function f is encrypted into E(f) by its sender, then E(f) is run on the potentially malicious site with x as an input. The result of E(f)(x) is returned and decrypted, yielding the result of f(x) [ST98].
[ST98] develops a function hiding method for polynomials. [LM99] presents a scheme based on error-correcting codes for matrices. Computations involving matrices can be hidden and all functions that can be represented by a matrix as well like, for example, combinatorial boolean circuits.

Conclusion

Mobile code is generally advocated for bringing substantially increased flexibility and extensibility (even if they have not been fully exploited yet). But these do not come without a price on security.

As we saw, a good level of host protection is achievable. There is currently a trend towards enabling finer grained access control schemes: thanks to this, mobile codes will probably find more useful applications in the near future. Semantics-based analyses might also provide a new means of securing hosts without compromising performances.

On the other hand, the protection of a mobile code against a malicious host is still an open research topic. Applying the theoretical solutions presented in this article to programming is far from trivial, and sometimes even unrealistic. Anyhow, it can be readily observed that non-cryptographic techniques are generally not sufficient to protect a mobile code.

References

[BMW98] Ingrid Biehl, Bernd Meyer, and Susanne Wetzel. *Ensuring the integrity of agent-based computations by short proofs*. In Kurt Rothermel and Fritz Hohl, editors, Proc. of the Second International Workshop, Mobile Agents 98, pages 183-194, 1998. Springer-Verlag Lecture Notes in Computer Science No. 1477.

[BY97] Mihir Bellare and Bennet Yee. *Forward integrity for secure audit logs*. Technical report, UC at San Diego, Dept. of Computer Science and Engineering, nov 1997.

[DFW96] Drew Dean, Ed Felten, and Dan Wallach. *Java security: From HotJava to Netscape and beyond*. In Proceedings of the 1996 IEEE Symposium on Security and Privacy, Oakland, Cal., May 1996.

[GJS96] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. Addison-Wesley, 1996.

[Gon98] Li Gong. *Secure java class loading*. IEEE Internet Computing, pages 56-61, november - december 1998.

[KAG98] G. Karjoth, N. Asokan and C. Gulcu. *Protecting the computation results of free-roaming agents*. In Kurt Rothermel and Fritz Hohl, editors, Proc. of the Second International Workshop, Mobile Agents 98, pages 195-207, 1998. Springer-Verlag Lecture Notes in Computer Science No. 1477.

[LM99] Sergio Loureiro and Refik Molva. *Function hiding based on error correcting codes*. In Manuel Blum and C. H. Lee, editors, Proceedings of Cryptec'99 - International Workshop on Cryptographic Techniques and Electronic Commerce, pages 92-98. City University of Hong-Kong, July 1999.

[LMP99] Sergio Loureiro, Refik Molva, and Alain Pannetrat. *Secure data collection with updates*. In Proceedings of the Workshop on Agents on Electronic Commerce - First Asia Pacific Conference on Inteligent Agent Technology, Hong-Kong, December 1999.

[Mic] Microsoft. *Authenticode WWW page*. http://www.microsoft.com/security/tech/authenticode/default.asp?id=21&parent=4.

[MvRSS96] Yaron Minsky, Robbert van Renesse, Fred B. Schneider, and Scott D. Stoller. *Cryptographic support for fault-tolerant distributed computing*. In Proceedings of the Seventh ACM SIGOPS European Workshop, pages 109-114, Connemara, Ireland, September 1996.

[Nec97] George C. Necula. *Proof-carrying code*. In Proceedings of the 24th ACM Symposium on Principles of Programming Languages, Paris, France, January 1997.

[NL98] George C. Necula and Peter Lee. *Safe, Untrusted Agents using Proof-Carrying Code*. Lecture Notes in Computer Science N. 1419. Springer-Verlag, 1998.

[ST98] Tomas Sander and Christian Tschudin. *Towards mobile cryptography*. In Proceeding of the 1998 IEEE Symposium on Security and Privacy, Oakland, California, May 1998.

[WSB98] Uwe G. Wilhelm, Sebastian Staamann, and Levente Buttyn. *Protecting the itinerary of mobile agents*. In 4th ECOOP Workshop on Mobility: Secure Internet Mobile Computations, 1998.

[Yee97] Bennet Yee. *A sanctuary for mobile agents*. Technical Report CS97-537, UC at San Diego, Dept. of Computer Science and Engineering, apr 1997.

[YY97] A. Young and Moti Yung. *Sliding encryption: a cryptographic tool for mobile agents*. In Proc. 4th International workshop fast software encryption 97. Springer-Verlag Lecture Notes in Computer Science No. 1267.