Mobile Agent Security[1]

Łukasz NITSCHKE[a], Marcin PAPRZYCKI[b], Michał REN[c]
a) Adam Mickiewicz University of Poznań, Poland
b) Warsaw School of Social Psychology, Warsaw, Poland
c) Adam Mickiewicz University of Poznań, Poland, and Institute for Infocomm Research, Singapore

Abstract. The aim of this chapter is to provide an overview of security issues facing mobile agent systems, and discuss ways of managing them. We explore the state of the art, to assess if it is mature enough for use in real-life security-critical applications like extraction of sensitive information, contract signing or broadly understood e-commerce.

1. Introduction to Mobile Agents

There exists number of scenarios where software agents are considered to hold promise for the future of computing. One of them is the vision of software agents utilized in the context of development and implementation of large complex systems [1]. Here, the benefits are grounded in basic principles of software engineering (i.e. decomposition, abstraction and organization) and include faster creation, easier maintenance, scalability, and an overall ease of deployment of complex distributed systems. Separately, agent approach is expected to play a crucial role when dealing with information overload [2]. Here intelligent software agents are to learn user preferences and act upon them in finding all and only such information that a given user is going to be interested in. Finally, software agents are also very often mentioned in the context of e-commerce, where they are to play a very important role in support of automatic price negotiations and purchase automation (see for instance [3] and references collected there). While there is no universally agreed upon definition of what a software agent is (in particular, there is *no formal definition*), most of existing ones are in some way similar to that put forward by Jennings [4]. There, agents are conceptualized as software artifacts that exhibit certain properties, such as:

- *autonomy* – they act without the need of constant human supervision,
- *sociability* – ability to interact with other agents or humans if necessary,
- *reactivity* – ability to react to changes in the environment ,
- *proactivity* – taking initiative, when appropriate, in order to reach objectives.

[1] Research reported in this chapter was supported by the KBN grant 0 T00A 003 23

This definition of software agents (as well as others) is very broad, and encompasses such programs as the animated paperclip in Word, many computer viruses, bots in first-person shooter games, auction agents in online auction sites, or search engines' web spiders. More interesting are so-called strong agents that, according to Jennings [5], have additional properties, such as: *veracity*, *benevolence* and *rationality* (however, the list of properties used to define software agents is much broader and definitely lacks consensus [6]). Finally, *mobility* – understood as agent ability to move from one computer to another, is believed to be particularly useful as it allows users to be offline (with their computer turned off) while their agents work on their behalf on other computers. It also serves to decrease network load – bandwidth-intensive tasks can be performed locally [7]. Moreover, agent mobility allows for load balancing and resource management in a grid-type environment, where agents are means of carrying across the network, and finding the best place to execute, computationally intensive tasks [8]. In this chapter, we focus on *mobile agents*, understood as agents conforming to Jennings' general definition, i.e. possessing autonomy, sociability, reactivity, proactivity, combined with mobility as an additional feature.

From the top-level perspective, mobile agent technology can be conceptualized as a highly distributed system which consists of two types of elements:

- *mobile agents* – programs that have the above mentioned properties and that, to fulfill their objectives, are equipped with a set of behaviors, which allow them to interact with the environment (including other agents) by exchanging messages or by acting within a local system (its own, or another);
- *mobile agent platforms* – middleware which:
  a) provide runtime environment (interpreter) for agent programs,
  b) allow agents to travel and communicate securely,
  c) offer various services.

Most common modern agent frameworks, such as JADE, FIPA OS or Grasshopper are implemented in Java. Such systems consist of an agent platform implementation and a programming interface, which:

- supports Agent Communication Language – ACL,
- enforces agents as containers for behaviors – while programmers only define behaviors.

Before proceeding further, let us address the question of usefulness of agent mobility, which is still being discussed with some researchers claiming that client-server type approach is more useful. While we have argued that at least in some scenarios there is a definite place for agent mobility ([7], [9]) settling this discussion is out of scope of this chapter. Therefore we assume that mobility is a useful property of software agents and therefore the question of *security of mobile agents has to be addressed*. Unfortunately, as this chapter shows, mobility is extremely challenging from the security standpoint. It may therefore be suggested that security of mobile agents may be one of important factors that prevents their actual use in real-life applications (for a discussion of issues involving agent system development, see [10]).

2. Security Requirements for Mobile Agent Systems

To discuss security requirements for mobile agents systems, let us take a closer look at a few possible applications of this technology (they are presented here as generic examples of context in which one can find mobile agents in the literature).

2.1. Typical Scenarios

2.1.1. Airfare Agent

It is often claimed that in the future mobile agents will be utilized to search the Internet for "the best offer," e.g. best available price of an airline ticket. In this case, we can easily imagine that we configure an agent and provide it with, at least a minimal necessary subset of the following travel defining data (but also possibly with other requirements):

- departure and destination points,
- date, maximum time of travel,
- class,
- list of preferred airlines,
- price threshold,
- credit card number, or electronic currency,
- digital signature key.

An agent constructed in such a way will be sent to visit airline web-sites (where the web-site does not necessarily mean only publicly available WWW sites, but may also include proprietary sites created to specifically handle such agents) and collect offers. We can also envision that since we provide our agent with information contained in the last three bullets above, we can authorize it to choose the best option, book the flight, and make the actual purchase. Furthermore, our agent may be able not only to collect posted prices, but also engage in price negotiations [11], [12].

2.1.2. Price Negotiators

More generally, mobile agents may act as price negotiators in a virtual e-market. They may move to the hosts and, acting on behalf of their users, engage in price negotiations (e.g. by participating in auctions) to win best prices and/or contract conditions [3]. In the case of auctions, to deliver the best possible results to their users, agents will have to use sophisticated strategies and will have to be precisely parameterized. Note that in some forms of price negotiations agents representing e-store will also have to use complicated negotiations strategies and their success will depend on them.

2.1.3. Network Management Agents

Finally, let us move away form e-commerce applications and consider agents working in a different field: monitoring and managing computer networks on behalf of network administrators. Assignments of such agents may include:

- installing, upgrading software on network devices and computers,

- monitoring network traffic, seeking illegal user activities and possible security holes,
- fighting worms and Trojan horses.

It is obvious that in each of the above listed scenarios, security of information stored "inside" of an agent is crucial to the system in which they operate. If the adversary knows, for instance:

- what is the threshold price – then he may modify his offer accordingly,
- what strategy is used by a given agent – then it may modify its own strategy and outwit it, or
- what technique is used to spot illegal users and their activities – then it may be able to modify its behavior in such a way that the guarding-agent will not be able to catch it.

Furthermore, security of the platform is also important as agents may try to subvert it in a variety of ways, similarly to what is currently done by computer viruses. Let us now look in more details into variety of threats that are possible in the context of mobile agent systems.

2.2. Classification of Threats

2.2.1. Platform-to-platform

Providing a secure transmission channel between platforms is the foundation of every solid mobile agent system. Properties of such channel should be as follows:

- *privacy* – agent migration (data and/or code) or messages exchanged by agents located on different platforms should remain secret; this need is obvious, since the mobile agent can carry e.g. a credit card number or a secret negotiation algorithm;
- *data integrity* – a traveling agent should be protected against malicious alterations of its data; for example somebody could try to erase some platforms from the path that is stored inside of the agent and that it is expected to follow;
- *authentication* – the source platform, the destination platform and agent's owner should be authenticated.

Obviously, the problem of secure communication in networks is well known, has been studied on its own right, resulting in many effective solutions.

2.2.2. Agent-to-platform

Let us consider the above presented air-fare agent scenario to indicate few possible threats to the security of the host (platform). An agent (sent by a competing airline, for instance) can spy on the host's databases, or disable services (e.g. performing a denial of service attack). In other words, it might act as a Trojan horse. To achieve this, agent may pretend to be legitimate, i.e. sent by a customer or, more seriously, to be a part of agent system of the given airline itself. Moreover, a benevolent agent may be corrupted by a third party, (captured en route in the case when the communication channel was not secure – see above) to act as a malevolent one and attempt to attack the platform.

2.2.3. Platform-to-agent

It is not only the agency that has to include safeguards against hostile agents; agents themselves are exposed to attacks by a hostile platform as well. For example, if the agency belongs to an airline, it might try to brainwash air-fare agents so that they forget all competing (presumably better) offers. Separately, if an agent is empowered to immediately accept an offer that is "good enough," the platform might try to disassemble it to find its threshold value and immediately make an acceptable offer. In general, the more the agent is authorized to do (while representing its owner), the greater the risk that at least one agency on its route will try to subvert it. For instance, we could have authorized the agent to make actual payments or to digitally sign contracts. In order to do that, the agent must carry with it a secret-token, such as the signing key, the credit card number, or even digital currency. It is not difficult to see that stealing one of these may be disastrous to the agent's owner, and very lucrative to the thief. Therefore, the main goals in protecting agents against threats posed by rogue platforms are as follows:

- *Privacy of computation* – which means that an agent is able to carry out computations without:
    - the host understanding what the agent is doing – computing with encrypted function (CEF),
    - the host being able to obtain agent secret data – computing with encrypted data (CED).

  This would allow the agent to carry secret values (signing key, e-money, credit card number) in an encrypted form but still involve them in necessary computations.
- *Integrity of computation* – gives a guarantee that the execution flow of agent code was not manipulated from outside of the agent.
- *Privacy and integrity of data* – assurance that data carried by the mobile agent has not been tampered with; this also applies to the agent's itinerary, which should be seen as part of the data.
- *Resistance to copy and replay*. Software agents have an inherent flaw – they can be easily copied and re-played. In this way one can simulate agent's environment changing it in controlled manner, trying to reverse engineer it, or at least find crucial parts of its functionalities e.g. determine the agent's price threshold.

2.2.4. Agent-to-agent

It can be argued that for all practical purposes an agent platform acts as an intermediary between interacting agents. Therefore, the problem of performing agent-to-agent attacks is actually a special case of agent-to-platform security. The following agent-to-agent attacks are possible:

- *impersonation* – agents pretending to be: other agents or agents of a certain owner,
- *denial of service* – agents preventing other agents from doing their job,
- *spying* – agents trying to steal secrets carried by other agents (embodied in their functions, or stored within data that they carry).

Separately, there exists also a group of illegal agent activities which cannot be controlled by the execution environment, like lying. Nonetheless, these have more to do with a broad question of trust, rather than strictly understood security (see below).

3. Cryptographic Goals and Tools for Mobile Agents

Security of every distributed system relies nowadays on some cryptographic techniques. Mobile agent systems are no exception, as security requirements of these systems overlap with main goals of cryptography. A standard list of cryptographic goals stated by [13] is as follows:

- *confidentiality* – keeping information secret from all but those who are authorized to see it,
- *authentication* – corroboration of the identity of an entity,
- *integrity* – ensuring information has not been altered by unauthorized or unknown means.

To achieve these goals cryptography provides us with many useful tools including:

- *hashing functions* – a computationally efficient functions that map strings of arbitrary bit length to some fixed length hash-values; hash functions are hard to invert and built in such a way that it is extremely difficult to find two different strings which yield the same hash-value;
- *Message Authentication Code* (*MAC*) *mechanism* – a set of hashing functions indexed by values from the set K – $H=\{h_k: k \in K\}$; here each $k \in K$ can be used to produce an authentication value of message m: $h_k(m)$ (MAC-value), which can be verified only by someone who knows k;
- *symmetric and asymmetric encryption systems* – consist of two types of transformations: E – encrypting and D – decrypting; every transformation is determined by a value called "the key;" in symmetric encryption systems keys for the inverse transformations ($D(E(m))=m$) are the same or trivially easy to compute, as opposed to asymmetric encryption where the keys are different and the decryption key is hard to compute from the value of the encryption key – in this case we speak of public (encryption) and private (decryption) keys;
- *digital signature schemes* – similar to asymmetric encryption, involve two types of keys and two types of transformations: S – signing transformations determined by private keys and V – verifying transformations determined by public keys; $S_s(m)$ – signature under message m created using key private s, can be verified by a public function (determined by a public key v) V_v which simply "says" whether the signature is valid or not.

While a number of methods is available to support cryptographic needs, their successful application to mobile agent systems is not easy. In the next sections we look in more detail into issues involved in cases described above. Note however, that the problem of secure communication described as platform-to-platform is exactly the same as the problem of providing security of any network communication (the fact that we are dealing with agent-to agent communication, or that we have agent migrating from one host to

another – which is also a case of communication security – does not make any difference). Since the case of network communication security is well known and there exist its many effective solutions we will omit it here and concentrate on the remaining problems that are specific to agent environments.

4. Mobile Agent Platform Security

Almost every PC user is aware of the fact that one should not run executable mail attachments for they may install a Trojan horse or spyware. In case of agents the situation is similar – to use an anthropomorphic description: an agent platform cannot be sure of agents' intentions. In the case of a rogue agent, it may even pretend to be legitimate, i.e. sent by a well-known customer or, more seriously, pretend to be a part of the local agent system itself. Moreover, a benevolent agent may be corrupted by a third party en-route to the server and, upon arrival, attempt to disrupt its operation.

There is a range of options available to safeguard hosts against such attacks, from simple to sophisticated [14], [15]:

- *Sandboxing* is one of the oldest methods of limiting resources available to mobile code, dating back to Java 1.0 and its Applet technology. Applets are small programs downloaded and executed by web browsers on a very limited Java Virtual Machine (sandbox), therefore preventing access to vital system resources. By default applets:
    - have no access to file system,
    - cannot communicate via network except with the originating host,
    - cannot start programs on the client side,
    - are not allowed to load libraries.

  Applets are a very nice technology as long as the application is limited – like an interactive online map – but in a number of instances, for example when code needs access to the file system (e.g. online virus scanner) simple sandboxing proves to be too inflexible.

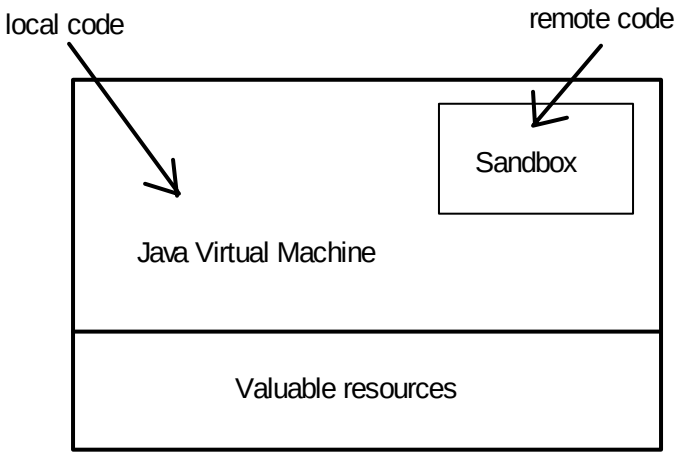

Figure 1 Sandboxing

- *Code signing* provides a method of distinguishing trusted and untrusted code using the mechanism of digital signatures. Trusted code can be granted access to critical system resources. Code signing was implemented, for instance, in Microsoft ActiveX and Java 1.1.

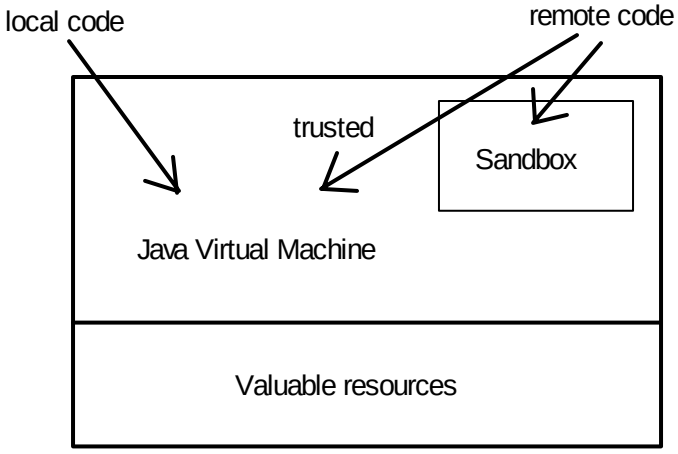

Figure 2 Code signing

- *Access control* – takes code signing one step further, by allowing the owner of the executing platform to precisely define security policies. Instead of dividing code into broad categories of "trusted" and "untrusted", code is associated with the identity of its owner and granted appropriate, individual access privileges. Access control was first implemented in Java 1.2. It allowed granting privileges to code signed by a certain identity. The security policy was described in special

configuration files – *access control list files*. One could specify detailed permission for a given originator, like:

- read/write/delete/execute permissions to a file or a subtree of a filesystem;
- connect/accept/listen permissions to host, domain, ip address, port;
- permissions to read environmental variables.

Java 2 introduced a new access control technology – Java Authentication and Authorization Services (JAAS). JAAS, finally, allowed Java Access Controller to grant privileges based on "who is running the code" (instead of "from whom the code is originating"). Unfortunately, all access control techniques impose a significant runtime overhead.

Separately, note that Java 2 security architecture does not fully protect against denial of service attacks as an agent is able to exploit platform's resources. This problem was solved in the Aroma Virtual Machine (AVM). AVM is a Java bytecode interpreter, which has built-in specific resource limitations mechanisms. One can specify for instance:

- agent's disk/network quota and transfer rates,
- allowed CPU usage.

Aroma became foundation of NOMADS mobile agent system [16], thus providing it with exactly the same amount of platform security as that available within the AVM itself.

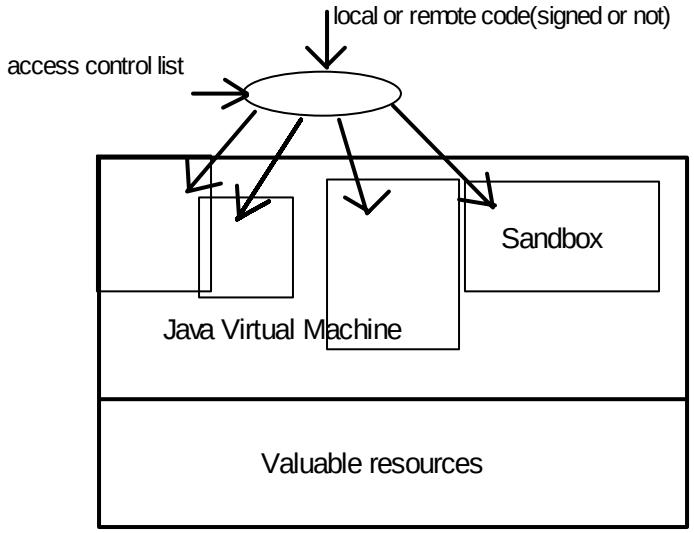

Figure 3 Access control

- *Proof-carrying code* – is claimed to be able to avoid expensive runtime checks, by performing code checking only once, before execution. In this method, code carries a proof of its good behavior. So far, this approach has not been implemented in practice and remains a purely theoretical one.

One can also imagine that instead of attacking the platform directly, agents try to attack other agents residing in it. For example, an agent may try to disable, destroy, or subvert agents of other customers, as well as agents that are a part of the agency infrastructure itself, e.g. the negotiation host [3]. However, protection against such threats is really just another aspect of platform security, as the agency has to protect all agents executing within it from one another the same way as it protects any other resource. Therefore, problems stated in sections 2.2.3 and 2.2.4 share the same solution.

Summarizing, it can be claimed that, while possibly generating substantial computational overhead, it is possible to protect agency against incoming rogue agents. The Aroma Virtual Machine and the NOMADS agent systems, while not reaching popularity of other agent platforms, have been a fully implemented and working proof-of-concept of this fact. It can be thus stated that the problem of securing the platform and agents from malicious agents can be considered solved, as the existing practical solutions are effective and any security breaches can only be caused by oversight, not fundamental flaws.

Let us now address the question of mobile agent security. We will focus on three aspects of this problem: privacy of computation, integrity of computation, and privacy and integrity of data (see: section 2.2.3, above) describing practical as well as theoretical solutions.

5. Mobile Agent Security

5.1. Privacy of Computation

5.1.1. Theoretical Solutions

5.1.1.1. Function Encryption

Computation of security critical function f in hostile environment may be done in an encrypted form. A natural way of encrypting a function is using composition with another invertible function g [17], [18]. Decryption is achieved by using g^{-1}: $g^{-1} \circ g \circ f$. There exists a class of functions which can be composed and inverted efficiently, namely rational functions (quotients of two polynomials). Moreover, the problem of finding rational function f when given only the composition $g \circ f$ is believed to be hard.

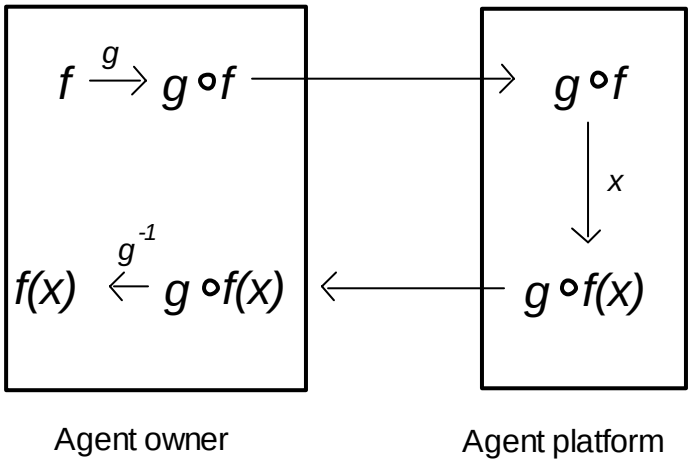

Figure 4 Function encryption

Authors of [17], [18] propose special digital signature scheme for mobile agents based on the idea of function encryption - *undetachable digital signatures (UDS)*. This type of signature mechanism consists of:

- *s* – signing transformation,
- *v* – verifying transformation,
- *r* – description of the requirements for messages which can be signed using *s*,
- *f* – function witch message *m* with requirements *r* (*f(m) = m||r*),
- $f_{signed} := s \circ f$.

UDS-enabled agent is equipped with *f*, f_{signed} which allow him to create a signature ($f(m), f_{signed}(m)$) under *m*. The signature is verified using *v* transformation as well as by confronting requirements *r* with *m*. Such scheme was presented in [19] and is based on RSA public encryption algorithm. Although agents can perform this special type of signatures, it should be noted that thus far function encryption cannot be applied, among others, to securing negotiation algorithms.

5.1.1.2. Homomorphic Rings

Privacy of computation can also be realized by employing two homomorphic rings R_1, R_2 [17], [18]. In order to ensure security we need a homomorphism $E:R_1 \rightarrow R_2$ which has the property of an encryption transformation. It means that *E* is hard to invert without knowledge of some secret information. *E* by definition should preserve ring operations ($E(x + y) = E(x) + E(y)$, $E(x * y) = E(x) * E(y)$), which would imply that all calculations in R_2 can be done on encrypted data. In practice it is hard to find such operation-preserving *E*. Therefore, we take an encryption function *E*, such that there exist efficient programs:

- *PLUS* – takes *E(x)*, *E(y)* and outputs *E(x+y)*,
- *MULT* – takes *E(x)*, *E(y)* and outputs *E(x*y)*.

Having these two basic operations we can securely compute every program *P*, which involves computing some polynomial $\Sigma a_{i1\ldots is} X_1^{i1} \ldots X_s^{is}$.

1. Agent owner:
    a) encrypts all input parameters: $E(x_1), E(x_2), \ldots, E(x_n)$,
    b) sends the parameters along with: *P, PLUS, MULT* to the agent platform.
2. Agent platform:
    a) encrypts all coefficients $a_{i1..is}$ in *P*,
    b) substitutes every +, * operation in *P* with *PLUS, MULT* calls respectively,
    c) executes *P* on encrypted parameters,
    d) sends the result to the agent owner,
3. Agent owner:
    a) decrypts $P(x_1,x_2,\ldots x_n)$.

It was shown that the procedure *MULT* may be replaced by *MIX-MULT*, but it requires a slightly different scheme than the one mentioned above. *MIX-MULT* calculates *E(x * y)* based on *E(x), y*. The following may serve as an example such system:

- $E: Z_{p-1} \rightarrow Z_p$, $E(x) = g^x \bmod p$, *g* is a generator of Z_p; *p-1* has small prime factors so that *E(x)* is can be inverted by somebody knowing *g*;
- *PLUS(E(x),E(y)) =E(x),E(y);*
- $MIX\text{-}MULT(E(x),y)=E(x)^y$.

An inherent disadvantage of this method is that it can only be applied to computing polynomials, while many practical security functions do not belong to this group. As a result, the contemporary knowledge on homomorphic rings does not solve the problem of providing privacy of computation.

5.1.1.3. Boolean Circuits

Every efficiently computable function on any number of input parameters $f(x_1, x_2, \ldots, x_n)$ can be represented as a Boolean circuit. There exist protocols that enable evaluation of such circuits in a distributed way, while keeping every participant unaware of all the inputs except for the ones belonging to him [20], [21]. The result may be either shared by all participants or every party may obtain only part of output. The heart of all such protocols is a concept of garbled circuits introduced by Yao [22]. The following diagram shows a sketch of a garbled circuit idea in a two-parameter case [23].

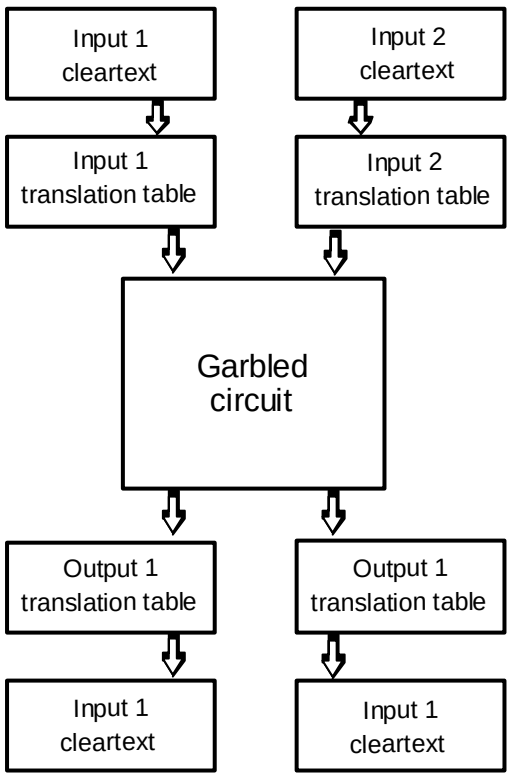

Figure 5 Garbled circuit evaluation

Let *GC* be a garbled circuit *C*. Distributed evaluation of such encrypted circuit *C(x,y)* requires the following steps:

1. A creates and sends the garbled circuit *C*:
    a) Encrypts *C* by assigning to signals {0,1} on every wire w_i in *C* a couple of unique keys k_i^0, k_i^1. Boolean functions performed by gates in *C* are substituted by a *garbled computation table* which maps input wires' keys to output wires' key(s).
    b) Creates *Input 1,2 translation tables* as the mapping of circuit input wires to corresponding keys chosen in a.
    c) Creates *Output 1,2 translation tables* as the mapping of circuit output wires to corresponding keys chosen in a.
    d) Sends
        1. keys representing bits of A's input x_1,
        2. Input 2 translation table,
        3. garbled computation table,
        4. Output 2 translation table.
5. B computes garbled circuit and obtains y_2:

e) Translates x_2 bits to respective keys.
f) Executes garbled circuit using: garbled computation table, A's input keys, and his input keys.
g) Translates his output bits to output 2 (y_2).
h) Sends A's garbled circuit output.

6. A:
   i) Translates output keys to y_1.

It is worth noting that garbled circuit masks actual signals on internal wires (has a black box property) and in consequence does not reveal any information about the input x_1. We can say that garbled circuit executed by B has x_1 hardwired.

Author of [24] attempts to apply the garbled circuit technique to mobile agents – as a safe way of evaluating security sensitive functions. Nonetheless, it has to be stressed that in [24] it was assumed that garbled circuits have to be created manually, so the crucial problem of creating automated garbled circuit compilers is not solved. Obviously, if garbled circuits cannot be created automatically then their usage in real-life situations will never materialize. Another issue is an obvious inefficiency of computing software as compared to hardware Boolean circuits.

5.1.2. Practical Solutions – Code Obfuscation

The most popular programming language for mobile agents is Java. Java source code is compiled into an intermediate code – bytecode. Bytecode is OS/machine independent, which gives it enormous portability and explains Java's popularity. However, bytecode can be easily decompiled and reverse engineered, unless code obfuscators are used. These tools scramble the bytecode making it difficult to analyze by employing the following techniques [25], [26]:

- layout obfuscation:
  - renaming identifiers (methods, variables, constants, types lose their original names),
  - removing debug information (code execution cannot be inspected in debug mode);
- control obfuscation:
  - altering execution flow by adding artificial branches, using conditional statements,
  - separating operations that belong together and mixing with other operations,
  - inserting redundant, meaningless code,
  - cloning methods – preparing different versions of a single method by applying various obfuscating techniques,
  - replacing method calls with inline code;
- data obfuscation:
  - splitting, merging and reordering arrays,
  - merging scalar variables,
  - converting static data to procedures,

- converting local variables to global.

A serious weakness of this method is the fact that it does not provide provable security; in fact, there is a constant "arms race" between obfuscators and disassemblers, and although it seems that so far the general-purpose disassemblers are outclassed by obfuscators, the situation may change radically as soon as specialized, deobfuscating disassemblers appear. Overall, it is clear that from an information-theoretic point of view, *obfuscation does not add to security at all* – its only function is to slow down the analysis of the algorithm.

5.2. Integrity of Computation – Theoretical Solutions

Known attempts to provide integrity of computation are based on holographic proofs and computationally sound proofs (CS-proofs) [27]. Here the trace of the execution shows not only the results, but also how they were obtained. Essentially, the preparation of such a proof consists of translating the claim (which must be formal, and self-contained) into an error-correcting form, and translating the proof. Any proof system (an algorithm that verifies a proof) can be reduced into a so-called domino problem, which is a graph-coloring problem. After that is done, verification takes the form of statistical checking of that coloring. The checking is very fast (in fact, it is polylogarithmic – faster then reading the proof), but has the probability of error of at most 50%. By repeating the checking, the probability of error can be arbitrarily reduced.

So far, these solutions remain theoretical. The main difficulty is the necessity of using formal logical systems. Even the simplest statements can become very complex when stated in a formal and self-contained way. In addition, the gain in speed over traditional proofs is only apparent when the proofs are large [28].

5.2.1. Privacy and Integrity of Data – Practical Solutions

An elegant solution which provides privacy and integrity of data was presented in [27] in the form of PRACs (Partial Results Authentication Codes). Every agent before leaving its home platform is supplied with a vector of keys. Every single key is used to create a MAC of the information gathered or computed on a certain server, and optionally to encrypt the data. The key is forgotten afterwards, preventing subsequent servers on the agent's path from tampering with gathered information. PRACs are used to preserve the integrity of dynamic data. Static, unchangeable data (i.e. agent's identity or itinerary), may be simply protected by a digital signature of the agent's owner. An attacker intending to change the agent's path without changing its identity would have to break the digital signature scheme. These two ideas of securing static and dynamic data were successfully implemented in Semoa agent platform [29].

Another, similar solution is giving the agent a public key, while the owner retains the private key. The agent may encrypt the information it collects with this public key, so that it can not decrypt it later. This ensures that nobody is able to cheat the agent, and pretend to be the home platform or the agent's owner. Unfortunately, public key cryptography is not as efficient as PRACs.

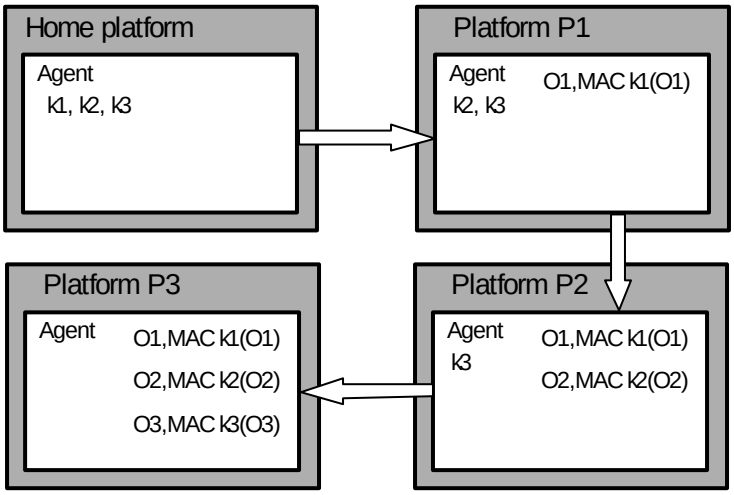

Figure 6 PRAC codes

Overall, it can be assumed that privacy and integrity of data can be assured using cryptographic techniques.

6. Trust-based Models

Let us now look from a completely different angle at the issue of agent system security. Let us start form a simple observation that some problems of agent security could become easier to solve if we make some reasonable assumptions about whom to trust, and to what extent, instead of being "uniformly paranoid" (see also [30]). This approach tries to address, among others, above mentioned questions of "lying agents" that try to gain advantage not by direct attack, but by pretending to be legitimate while trying to disrupt the system by various forms of malicious disruptive behavior (e.g. agents that in certain forms of auctions cause the final price to be higher then the true valuation of the product [31]).

6.1. Hardware-based Solutions

The most far-reaching assumption would be to trust the agent platform. However, even if the company maintaining the agent platform is trustworthy, the software may be compromised by an outside adversary. A partial solution to this danger is the use of Hardware Security Modules (HSMs) [32]. A platform can have a secure hardware device that cannot be tampered with, has a very restricted set of inputs and outputs, and is capable of performing cryptographic operations. Such a device can then be involved in interactions between the agent and the agency, ensuring for instance that they are performed only once. HSMs also typically include secure clocks (so that e.g. transaction time can be recorded

accurately), hardware-based random number generators, and tamperproof key storage space. They also offer very high cryptographic performance – up to two orders of magnitude faster than software solutions. Unfortunately, HSMs are expensive (thousands of dollars each), and their use implies trust in the maker of that HSM (which comes back to the old riddle: who controls the controller? – who will assure that creators of such modules can be trusted and did not leave secret backdoors in their modules, e.g. known to the government agencies).

A much cheaper solution that is gaining popularity is Trusted Platform Module (TPM) – a chip that can be included on a standard PC motherboard, and shares some similarities with HSMs. However, for agent platforms, the most useful new ability is attestation of the platform. [33] As soon as the computer boots, the TPM chip starts gathering platform metrics, storing those metrics in the log, creating hashes of those metrics and storing the hashes in the Platform Configuration Registers (PCRs). The PCRs can then be signed using the TPMs Attestation Identity Key (AIK) at some point. This certifies that the computer is running certain software, so that the remote user (or agent) can be sure that this is indeed the case, and that the software has not been maliciously modified, since the Platform Configuration Registers can not be arbitrarily set; they can only be reset or extended. The agent can be constructed in such a way, that its secrets are "sealed" – only revealed on a platform meeting certain requirements. While TPM does not completely solve the platform trust problem, it could make cheating on part of the platform owner more difficult, and encourage trust in agent platforms.

6.2. Cooperating agents

In most cases it makes sense to limit trust to platforms, no matter what assurances they offer. There are a few interesting attempts to address the platform trust problems by making the agents cooperate. This approach makes an assumption that not all servers are malicious and not all of the corrupted ones want to collaborate with one another. It seems reasonable in e.g. a network of competing companies. Therefore, it is vital that the cooperating agents are scattered among the system nodes rather than located on the same server.

Roth in [34] proposes a scenario, in which agents work in pairs – let us call them agent *A* and *B*. *A* visits a set of hosts *H*, while his partner moves to a server which belongs to a rival of *H*. *A* walks the predefined path and passes the offers to *B*. Once *A* has collected all the information they can choose the best offer; moreover, they can pay for it using e-money shared by them using a secret sharing scheme – *B* can send his part of the e-coin.

Authors of [35] show that we can attempt to build self-supporting communities of agents, so that every member of the community has at least two guards of its security – the Shared Security Buddy Model. Somewhat similarly, in [36] a trust-based security model for agent communities was presented. It was shown how it is possible to sustain long-term *soft security* – defined as a situation where isolated cases of mischief are possible, but in a long-term system will adapt its behavior and eliminate offending agents. While there exist a number of application areas where soft-security can be sufficient (e.g. within a "closed" organization, where soft security measures are supported by other inter-organizational security measures; or when security is only of limited true concern – see next paragraph), it

is clear that this level of security is not enough in the case of e-commerce, or when dealing with any type of sensitive information management in an open system. Observe that for a cheat it is enough to score once – steal large sum of money and disappear. In this case, the fact that he would have been eliminated over time from the system is not good enough for these who lost their money, or whose sensitive information was compromised.

Certain real-world situations lend themselves naturally to creation of a network of agents which, while not cooperating per se, are able to communicate, and would be more resistant to corruption. Consider, for example a network of personal assistants, which all keep track of CD's that their owners like. In order to get a recommendation, one could have his assistant question assistants of people who liked similar CD's. Even if a fraction of agents maliciously cheat (for example to promote a CD), the net effect would be mitigated by the honest ones. Unfortunately, since creation of agents is "cheap," such networks are susceptible to corruption by masses of "special agents" acting in concert. Possible solutions are: accepting new members only by invitation (which defeats the purpose of open exchange of information), creating trust-networks or ensuring that a human spends time before releasing next agent, by giving out a test that only a human can pass [37]. Such tests, called CAPTCHAs, are based on unsolved problems in artificial intelligence – usually image or speech recognition. Nevertheless, they will not deter a determined adversary – they just increase the cost of introducing rogue agents. Note however, that for recommender systems (even these involved in e-commerce), the fact that someone would be convinced to buy a CD would not have "disastrous" consequences. This is precisely the type of scenario, where approaches similar to these proposed in [36] would provide an acceptable level of agent-system security.

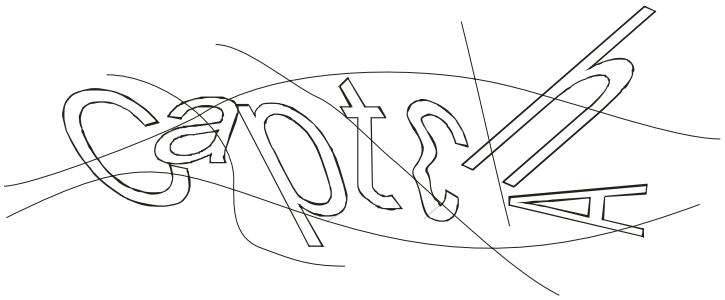

Figure 7 Sample CAPTCHA

6.3. Undercover agents

Honesty of the platforms can be verified in the same way as corruption of organizations in the real world – by undercover agents. An agent may pretend to represent a customer, and search for the best air-fare price, but it might have been prepared to contain certain data, that should never change, if the agencies are honest. If the data does change, one can be sure that at least one agency on the agent's path has acted malevolently. It is then easy to isolate the rogue agency by using more such agents with different paths.

6.4. Clueless agents

Finally, if no platform is to be trusted, it is possible to create agents that do not know their intended purpose [38]. An example would be an agent that performs a patent search by calculating hashes of strings, and trying to match them with a stored value. The owner of the agent prepares it beforehand by calculating:

N := a random nonce
K := H(description of the patent idea)
M := E_K(action to be performed upon finding the idea)
O := H(N $\oplus$ description of the patent idea)

The agent then searches through the database, hashing the strings it finds and testing if

H(N $\oplus$ tested string) = O, and
if yes, the agent executes $D_{H(\text{tested string})}(M)$

Therefore, if a certain string is in the database, it will be found, but one cannot derive it from the agent beforehand, so the patent idea stays safe. Likewise, it is impossible to know what the agent will try to do once the string is found. Note that this approach may give rise to new type of hard to defeat viruses and worms, which will not reveal their payload until they infect a system with a certain domain name, or a certain item appears for sale on eBay. Unfortunately, clueless agents tend not to be very efficient, and not every problem can be solved by them. In particular, as agents operate blindly, they have to search through a very large number of possible solutions before stumbling upon the correct one. In this case it is exactly the cluelessnes that precludes use of any optimization.

7. Concluding remarks

In this chapter we have presented an overview of security issues involved in mobile agent systems. We have established that problems involved in agent communication and security of an agent platform can be considered as practically solved. Obviously, the same way as network communication is secure only until a more powerful hacking method is developed (which is then counteracted by a new security measure), sandboxing the platform will be effective until someone finds a hole in the virtual machine (that will later have to be patched). Nevertheless, we consider the agent platform and agent communication as relatively safe. Unfortunately, as our research shows, none of the existing methods can guarantee true agent security. Section 5 shows that only data carried and collected by mobile agents can be efficiently secured. It means that contemporary knowledge of agent protecting techniques restricts us to “window-shopping” type of mobile agents. Broad utilization of mobile software agents in realistic scenarios remains a question of future inventions. However, our research indicates also that when only soft security is required, and when response time to an existing threat is not crucial, communities of cooperating agents can eliminate bad agents from the system. We can thus say that it is possible to create weakly secure self-securing agent systems.

References

[1] Jennings N. R. (2001) *An agent-based approach for building complex software systems*, Communications of the ACM, 44 (4), pp. 35-41

[2] Maes P. (1994) Agents that Reduce Work and Information Overload. Communications of the ACM, 37, 7, pp. 31-40

[3] Ganzha M., Paprzycki M., Pîrvănescu A., Bădică C., Abraham A. (2004) *JADE-based Multi-agent E-commerce Environment: Initial Implementation*, Analele Universităţii din Timişoara, Seria Matematică-Informatică, Vol. XLII, 79-100

[4] Jennings N. R., Wooldridge M. (1998) *Applications Of Intelligent Agents*, Springer-Verlag NY, Agent technology: foundations, applications, and markets, pp. 3-28

[5] Jennings N. R., Wooldridge M. (1995) *Intelligent Agents: Theory and Practice*. The Knowledge Engineering Review, pp. 115–152

[6] Galant V., Tyburcy J. (2001) *Intelligentny Agent Programowy*, Prace Naukowe AE Wrocław, Nr 891, pp. 46 – 57, in Polish.

[7] Bădică C., Ganzha M., Paprzycki M. (2005) *Mobile Agents in a Multi-Agent E-Commerce System*. In: Proceedings of the SYNASC 2005 Conference (to appear)

[8] Di Martino B., Rana O.F.(2003) *Grid Performance and Resource Management using Mobile Agents*, in: Getov, V. et. al. (eds.) Performance Analysis and Grid Computing, Kluwer, 2003

[9] Bădică C., Ganzha M., Paprzycki M. (2005) *UML Models of Agents in a Multi-Agent E-Commerce System* In: Proceedings of the ICEBE 2005 Conference, IEEE Press, Los Alamitos, CA, 56-61

[10] Paprzycki M., Abraham, A. (2003) *Agent Systems Today: Methodological Considerations*, Proceedings of the 2003 International Conference on Management of e-Commerce and e-Government, Jangxi Science and Technology Press, Nanchang, China, pp. 416-421

[11] Bădică C., Bădită A., Ganzha M., Iordache A., Paprzycki M. (2005) *Implementing Rule-based Mechanisms for Agent-based Price Negotiations*. In: Proceedings of the ACM SAC Conference (to appear),

[12] Bădică C., Bădită A., Ganzha M., Iordache A., Paprzycki M. (2005) *Rule-Based Framework for Automated Negotiation: Initial Implementation*. In: Proceedings of the RuleML Conference (to appear)

[13] Menezes A., van Oorschot P., Vanstone S. (1996) *Handbook of Applied Cryptography*, CRC Press, CRC Press, Boca Raton, USA

[14] Loureiro S., Molva R., Roudier Y. (2000) *Mobile Code Security*, Proceedings of ISYPAR 2000 (4ème Ecole d'Informatique des Systèmes Parallèles et Répartis), Code Mobile

[15] Dageforde M. *Security in Java 2 SDK 1.2* , http://java.sun.com/docs/books/tutorial/security1.2/overview/

[16] Suri N., Bradshaw J., Breedy M., Groth P., Hill G., Jeffers R., Mitrovich T. (2000) *An Overview of the NOMADS Mobile Agent System*, in Proceedings of ECOOP 2000

[17] Sander T., Tschudin C. (1997) *Towards Mobile Cryptography*, International Computer Science Institute technical report 97-049

[18] Sander T., Tschudin C. (1998) *Protecting Mobile Agents Against Malicious Hosts.* Lecture Notes in Computer Science 1419, pp. 44

[19] Burmester M., Chrissikopoulos V., Kotzanikolaou P. (2000) *Secure Transactions with Mobile Agents in Hostile Environments*, Proceedings of the 5th Australasian Conference on Information Security and Privacy table of contents, pp: 289 - 297

[20] Tate S., Xu K. (2003) *On Garbled Circuits and Constant Round Secure Function Evaluation,* Computer Privacy and Security Lab, Department of Computer Science, University of North Texas, Technical Report 2003-02

[21] Beaver D., Micali S., Rogaway P. (1990) *The round complexity of secure protocols*, Proceedings of the twenty-second annual ACM symposium on Theory of computing, pp. 503-513

[22] Yao A. (1986) *How to Generate and Exchange Secrets*, In 27th FOCS, pp. 162-167

[23] Lindell Y., Pinkasy B. (2004) *A Proof of Yao's Protocol for Secure Two-Party Computation*, Electronic Colloquium on Computational Complexity, Report No. 63

[24] Lien H. (2002) *System Design and Evaluation of Secure Mobile-Agent Computation with Threshold Trust*, http://zoo.cs.yale.edu/classes/cs490/02-03a/lien.henry/

[25] Hongying L. (2001) *A comparative survey of Java obfuscators available on the internet*, http://www.cs.auckland.ac.nz/~cthombor/Students/hlai/

[26] Collberg Ch., Low D., Thomborson C. (1997) *A Taxonomy of Obfuscating Transformations*, http://www.cs.arizona.edu/~collberg/Research/Publications/CollbergThomborsonLow97a/

[27] Levin L. (1999) *Holographic Proofs*, http://www.cs.bu.edu/fac/lnd/expo/holo.html

[28] Roth V. (2002) *Empowering mobile software agents*, Proceedings of 6th IEEE Mobile Agents Conference, Suri N, ed., Lecture Notes in Computer Science, vol. 2535 pp. 47–63

[29] Yee, B. (1997) *A sanctuary for mobile agents*. Technical Report CS97-537, Department of Computer Science and Engineering, UC San Diego,

[30] The Trusted Computing Group (2004) *TCG Specification Architecture Overview*, https://www.trustedcomputinggroup.org/groups/infrastructure/

[31] Garg, N., Grosu, D., Chaudhary V. (2005) An antisocial strategy for scheduling mechanisms, Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium.

[32] Hardware Security Modules, history: http://www.eracom-tech.com/pioneering.0.html#1526

[33] Bajikar, S., Trusted Platform Module Whitepaper, Intel Corporation, Mobile Systems Group, June 2002, http://developer.intel.com/design/mobile/platform/downloads/Trusted_Platform_Module_White_Paper.pdf

[34] Roth V. (1999) Mutual protection of co-operating agents, Vitek J., Jensen C. eds., Secure Internet programming: security issues for mobile and distributed objects, Lecture Notes In Computer Science vol. 1603, pp. 275-285

[35] Page, J., Zaslavsky, A., Indrawan, M. (2004) *A Buddy Model of Security for Mobile Agent Communities Operating in Pervasive Scenarios*, Proceedings of Second Australasian Information Security Workshop (AISW2004), pp. 17-25.

[36] Hexmoor H., Bhattaram S., Wilson S. (2004) *Trust-based Security Protocols*, SKM 2004 Workshop, SUNY Buffalo, September 2004

[37] von Ahn L., Blum M., Hopper N., Langford J. (2003) *CAPTCHA: Using Hard AI Problems for Security*, Advances in Cryptology, Eurocrypt 2003

[38] Riordan J., Schneier B. (1998) *Environmental Key Generation towards Clueless Agents*, Mobile Agents and Security, G. Vigna, ed., Springer-Verlag, pp. 15-24.