Secure Network Management within an Open-source Mobile Agent Framework

A. Pashalidis and M. Fleury

Department of Electronic Systems Engineering
University of Essex
Wivenhoe Park
Colchester, UK
Tel.:+44-(0)1206-872817 – Fax.: +44-(0)1026-872900
e-mail: {apasha,fleum}@essex.ac.uk

Secure Network Management within an Open-source Mobile Agent Framework

Abstract

Mobile agents (MAs) have been proposed for de-centralized network management. This paper explains how Aglets, a Java open-source MA framework, not a proprietary system, can be used for secure network management, offering an alternative to or complementing SNMPv3 security. The solution prototyped is a hybrid environment where network management applications use MAs to interact locally with SNMP agents via the SNMP protocol. The implemented class libraries extend the security infrastructure of Aglets, by incorporating cryptographic functions through the Java Cryptography Extensions (JCEs). The extension enables data fields to be encrypted, while code is to be digitally signed. Legacy SNMPv1 and v2 enabled devices, with elementary security, can also be upgraded through this approach, which can effectively avoid a range of attacks. Consideration has been given to auxiliary functionality such as responding to SNMP traps, key distribution, logging, and secure clock synchronization.

Keywords: Mobile agent, network security, SNMP, aglets

Suggested Running Head
Secure Open-Source Mobile Agent Framework

1 Introduction

In this paper, we consider mobile agent security, which has the ability to remedy network security vulnerabilities and for legacy devices might make it unnecessary to upgrade to SNMPv3 [1, 2], introduced in 1998, which indeed does include enhanced security features. In other cases, mobile agents can provide a distributed service for those network management activities where it is appropriate, complementing SNMPv3's centralized service. The mobile agent variant considered, aglets, can avoid firewall blocking.

Management applications acquire information about the network constantly through enquiries to network devices such as routers, switches, hubs, uninterruptible power systems, printers, workstations and various servers. When a problem arises, the application software can notify the network manager in "real time". Using management applications such as OpenView [3], NetSP, and SunNET [4], network administrators are assisted in their work. Management activities may include device monitoring, performance measuring, problem isolation and resolution, growth planning and quality of service (QoS) assurance, in other words a subset of the ISO FCAPS functional areas [5]. Since its development in 1987 and standardization in 1990, IETF Simple Network Management Protocol (SNMP) [6], which is an application layer protocol in the TCP/IP suite, has become the message request protocol of choice[1], deployed in many companies' IP intranets, and in the network core. Unfortunately, SNMP implementations have two types of security weakness [7]: 1) Network insecurity 2) Message and trap handling [8], which have led to a recommendation that this service is blocked at perimeter firewalls, those that monitor egress as well as ingress traffic [9].

---

[1] The OSI Common Management Information Protocol (CMIP) [5] follows a similar client/server interaction model to SNMP, but its memory requirements are large. Its four-year ITU-T standardization cycle impeded its acceptance.

Common network vulnerabilities are well known and largely date from an era when network security was not a strong issue. In SNMPv1 and v2, the password or community string is passed in plaintext to the device. In some installations, the community string defaults for read-only access, "public", and read-write access, "private", are not altered. There exist scanners to locate devices running SNMPv1 such as SNMPing and SNscan, making it possible that the community string can be altered, and hence denying access to the owner of the device[2]. Because SNMP commonly uses UDP (originally for robustness), which is connectionless, firewalls (if access to ports 161 and 162 *inter alia* is not filtered out) cannot establish whether the source of an original request came from within the managed LAN. Responses to service requests, and traps (notifications of unusual events) are respectively made to/from the originating unprivileged port (again problematic to monitor) without any connection setup and hence without establishing random sequence numbers.

Most versions of SNMP are based on the client/server model where the network management station (NMS), hosting the management application console, is the client and the network devices play the role of servers. A static agent on the server acts as a proxy[3], servicing requests made upon the Management Information Base (MIB). In the basic SNMP architecture, the NMS can become a network bottleneck [10] as a result of the polling-based nature of monitoring and data collection, which can also increase the latency of reporting for larger networks. There are problems with potentially restricted message sizes and an overshoot effect if bulk transfer is utilized (in SNMPv2). SNMPv3 does allow data compression by adding encryption envelopes [11] and has belatedly introduced a form of management delegation [12], but the main emphasis of SNMPv3 remains security and not scalability. Nevertheless, for management applications deployed on larger networks, lack of scalability remains an issue,

---

[2] Scanners also allow the network administrator to locate any SNMP devices in order to disarm potential attacks.
[3] The proxy arrangement allows devices without a TCP/IP stack such as some bridges or modems, or non TCP/IP networks to support SNMP. This paper does not consider alternative SNMP architectures such as when an RMON probe is placed between NMS and agent, or when the NMS is fronted by an agent.

leading to a number of proposals for decentralized solutions including distributed object management [13], web-based management through push management [14], and Directory Enabled Networks [15].

However, centralized solutions also have other disadvantages such as the need for the operator to physically direct data acquisition, and the semantic simplicity of the data requested [16]. Mobile agents (MA) for network management are a further proposal that do not require abandonment of SNMP-enabled devices but may contribute to scalability and flexibility. In the MA paradigm, remote programming occurs by virtue of mobile code. The MA can perform micromanagement operations at the network device, before consolidating the results of enquiries and returning to the NMS.

If MAs journey between a number of devices before returning to the NMS, collecting data on the way then congestion may be transferred elsewhere. Still, hierarchical organization of MA management [17] offers a solution to this problem. Another problem that MAs can bring is an increase in response time, but by confining MA operations to applications that require frequent testing of many MIB variables, with subsequent computation upon these, while retaining centralized enquiries to requests to just a few variables, for example testing the state of a link, then this weakness can also be addressed. In fact, an MA can perform in polling or iterative mode. Resource reservation after processing MIB data has been identified as a suitable application for MAs [18] acting in decentralized fashion.

The main impediment to adoption of MAs is the need for strong assurance that network security will not be compromised as, in historical terms, an MA has similarities with a virus. One principal of network security [19] is that proposed solutions should be open to inspection and comment by domain experts. In particular, security should not rely on concealment of the algorithm. A recent example illustrates the problem: the breaking of the Wired Equivalent Privacy algorithm of the 802.11 wireless LAN standard [20]. The purpose of the current paper is

not to demonstrate a complete solution but to show how a secure solution can potentially be produced with existing software technology.

Mobile code requires interpretation rather than compilation, which has the added advantage that the interpreter software footprint is small, making it suitable for embedding in network devices. The Java Virtual Machine (JVM) has a stack-based architecture that reduces the size of mobile code. As is well-known, JVM implements a security `sand-box' [21] guarding against access to unauthorized resources. Bytecode verification in Java can, for instance, catch buffer overflows through code modifications. Some issues connected with type preservation and soundness present in Java 1.1 [22] apparently have been addressed in Java 1.4. Therefore, this paper builds on an existing general purpose MA system, Aglets [23], but through a set of class libraries extends the security infrastructure of Aglets, by incorporating cryptographic functions from the Java Cryptography Extensions (JCEs), the API of which is now an in-built feature of Java version 1.4. Though the proposal to use MAs for network management is not new (Section 2), hitherto, some systems have been special-purpose and/or reported security features at a high-level. A weakness of special-purpose systems is that they are not necessarily extensible or open.

The rest of the paper is organized as follows. Section 2 is a brief review of MA systems and network management, particularly in regard to security issues. Section 3 presents the design of the hybrid MA system introduced in this paper, and includes a discussion of its security compared to pure SNMP systems. In Section 4, the core security arrangements are detailed, whereas in Section 5, auxiliary security functionality is considered. Finally, Section 6 provides a summary of the paper.

2 Background

2.1 Some MA network management systems

JAMES [24] is a Java-based platform of mobile agents for the management of telecommunication networks. The network management application runs at the JAMES manager, every network

element runs a JVM and an MA server (called "Agency"), and development of code is separated from the running environment via a dedicated code server. However, despite many innovative features, the JAMES platform does not so far [24] consider security aspects of the system and does not provide authentication or encryption schemes. In [25] integration of SNMP and mobile agents is examined and SNMP-enabled mobile agents are introduced. An extension to the SNMP, DPI (Distributed Protocol Interface) protocol is proposed in order to enhance the interaction of mobile agents with SNMP agents, but security aspects, are not covered. The AMETAS (Asynchronous MEssage Transfer Agent System) [26] is used and extended to support SNMP operations. There are some security mechanisms incorporated into the AMETAS system, as access to local resources is achieved through services. This means that agents do not gain direct access to system resources, but have to pass through an appropriate service structure. As reported in [26], AMETAS has possibilities for authentication, access control, and encryption but these aspects are mentioned in passing.

The Mobile Agents for the Management of Applications and Systems (MAMAS) environment [27] does explicitly consider security. MAMAS gains its openness by virtue of compliance with the OMG Mobile Agent System Interoperability Facility (MASIF). Access security policies are stored in encrypted files. Authentication is addressed through the DSA (Digital Signature Algorithm) [28] and X.509 certificates. Cryptography provides security for agent transfer and message exchange, either through symmetric DES keys (exchanged by an RSA mechanism or a Diffie-Hellman exchange) or indirectly through the SSL (Secure Socket Layer) protocol. The current paper, supplies some of the implementation detail that, as primarily a design paper, by necessity was avoided in [27], taking advantage of subsequent security libraries, and providing a solution that is directly implementable by others.

2.2 The Aglets framework

The Aglets framework, originally developed by IBM, has now become open-source and is publicly available [22]. Mobile agents in the framework, aglets, have a weak form of mobility [29]: as a Java object, an aglet's code and data variables are serialized but not the execution stack for obvious security reasons. One programmability weakness of Aglets is that there is a single re-entry point [30] on arrival at a remote context. The framework includes a context server, called Tahiti. Aglets are created in Tahiti, transferred between Tahiti servers via the Agent Transfer Protocol (ATP), cloned, (de)activated and disposed of in Tahiti. ATP is layered upon TCP using a default but unprivileged port number. As the Aglet Framework is built upon the JVM, a practical deployment scenario requires that every network device in the managed network hosts a JVM. In [31], JVMs were integrated into a family of network devices from Nortel and the value of the approach was demonstrated.

When a corporate firewall blocks all access except to the HTTP well-known port(s) (80 or 8080), or for some other reason such as the use of unprivileged ports, causes blocking, then ATP can employ HTTP tunneling. The Aglet API provides a `retract' mechanism whereby a client within a firewall can recall an aglet, rather than the aglet make a new TCP connection to the NMS from the firewall exterior. In fact, retraction was not employed in the system considered herein as it circumvents a security measure even though some authorities, *e.g.* [32], consider firewalls of limited value.

2.3 MA security issues

In general, the issue of mobile agent security, including the issue of protection against a malicious host, has been given considerable coverage (not considered in this paper, and the reader is referred to the volume in which the following reference [33] appears. Some implementations are briefly reviewed herein. One point *not* generally of concern, as it might be in e-commerce, is interaction between agents launched by independent users. Though agent cooperation is possible in SNMP servicing, in this paper inter-agent messaging is not part of the core system.

The Java-based Ajanta system [34] actively addresses some of the security related aspects of mobile agents. Ajanta offers a mechanism for authenticated communications, whenever an agent is to be transferred between two servers. This mechanism relies on an application layer challenge-response protocol, which employs nonces to guard against replay attacks. Otherwise, the agent host is protected through the resource monitoring and access control services of Ajanta, which rely on the corresponding Java security constructs such as the Access Controller, the Security Manager, the Class Loader and the Thread Group mechanism, likewise employed in the

current system. Transmission protection of agents in Concordia [30] is entirely based on the Secure Sockets Layer version 3 (SSLv.3). No encryption, signing or authentication mechanisms are incorporated at higher levels in Concordia, since these are already encapsulated into the SSL layer. In particular, Concordia does not make use of code signing since, according to [30], "it does not supply a suitable basis for the security of a mobile agent environment". In this respect, refer to Section 4.1. D'Agents [35], an agent system that cannot necessarily exploit Java security, makes use of PGP (Pretty Good Privacy) [36] to generate asymmetric RSA keys, though this results in a reduction in efficiency, as PGP is not tightly integrated into the system.

General security issues in Aglets [37] have also been addressed. By means of an aglet proxy, control over method access is exerted. An aglet can have a policy attached, which, for example, regulates the number of hops, and the lease time on each host. Aglets are given a unique and random identifier upon creation that might be used in a policy to refer to a particular aglet. In addition, the Aglets framework avails itself of all the Java security constructs such as the Bytecode Verifier, the Class Loader, the Security Manager and the Access Controller [21].

Though the Concordia approach of simply adopting an existing cryptographic standard, with no alternative, is possible, this would not be an extensible approach, whereby the programmer could substitute cryptographic algorithms (though this activity itself presents a risk). SSL incurs an overhead due to the key agreement protocol. Additionally, the open source version of aglets does not presently support SSL.

3 Aglet and SNMP system

### 3.1 Integration with SNMP

In [24, 25, 26] the integration of mobile agents with SNMP is considered. The advantage of such a hybrid system is that it is backwards compatible with existing SNMP management applications. The structure of the current hybrid approach that combines mobile agents and SNMP in network management is illustrated in the Figure 1.
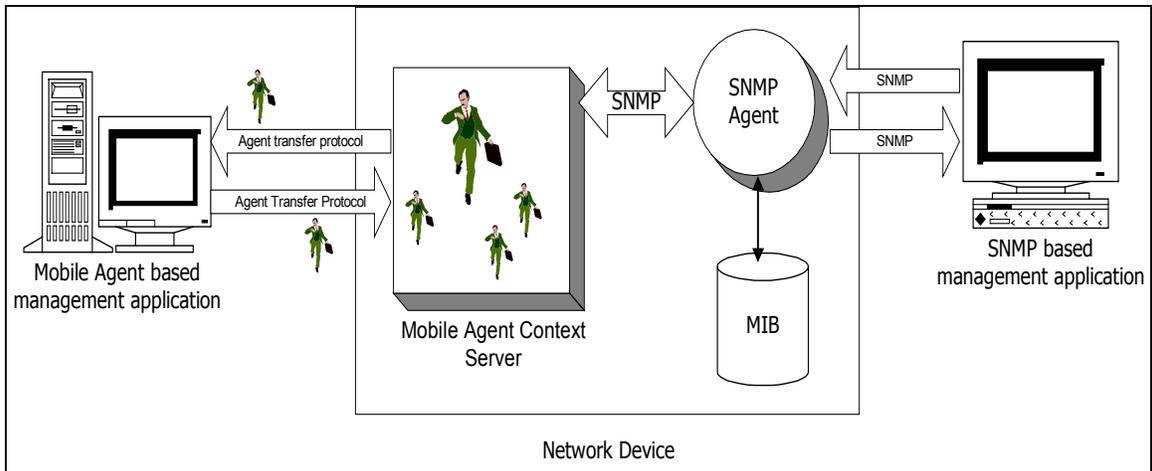
Figure 1: Hybrid network management system combining mobile agents and SNMP

From Figure 1, every mobile agent enabled network device has to offer a mobile agent context server. The mobile agents hosted in the context communicate with the local SNMP agent via SNMP, just like conventional SNMP based management applications. A network can be managed by MA-based management stations as well as stations that rely on SNMP.

The management application creates mobile agents that it then 'sends out' to the network to perform their task autonomously. Each agent has a specific task and an itinerary. The itinerary is a list of network devices the agent is to 'visit' and perform its task. The agent returns afterwards and sends a report to the management application. Typically, it is disposed of directly afterwards, as sending that report would conclude its lifecycle. The management station-side of the architecture is illustrated in Figure 2.
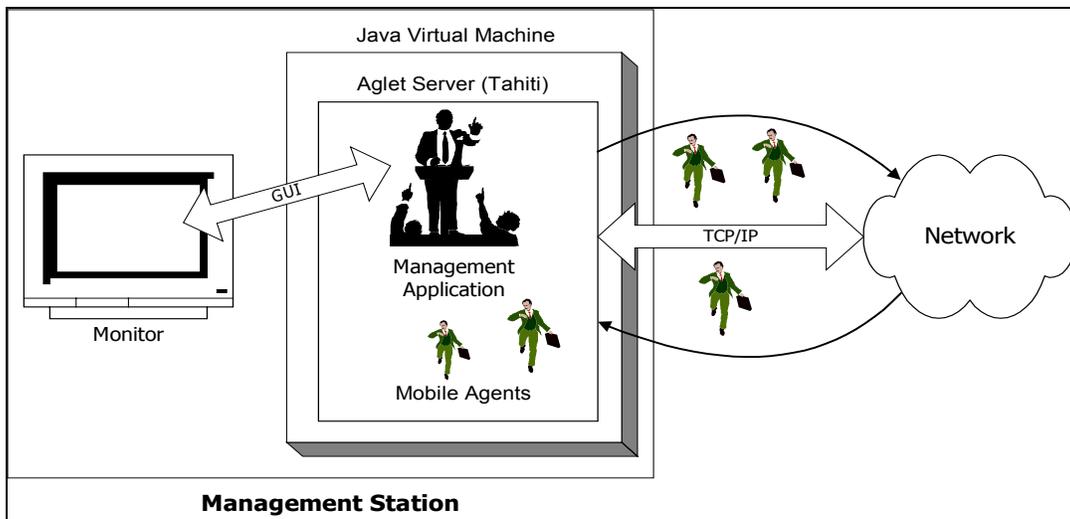
Figure 2: Management station side of implementation

As Figure 2 implies, the management application was chosen to be a (stationary) aglet itself. This decision was made just because it allows it to easily spawn and communicate with aglets and provide a GUI at the same time. In principle, the management application can be a stand-alone application as long as it is able to spawn and communicate with Aglets. Mobile agents follow their itinerary autonomously, performing the required tasks at the remote network devices, and return afterwards with the accumulated information and 'report' it to the management station. The network device side of the implementation is illustrated in Figure 3.
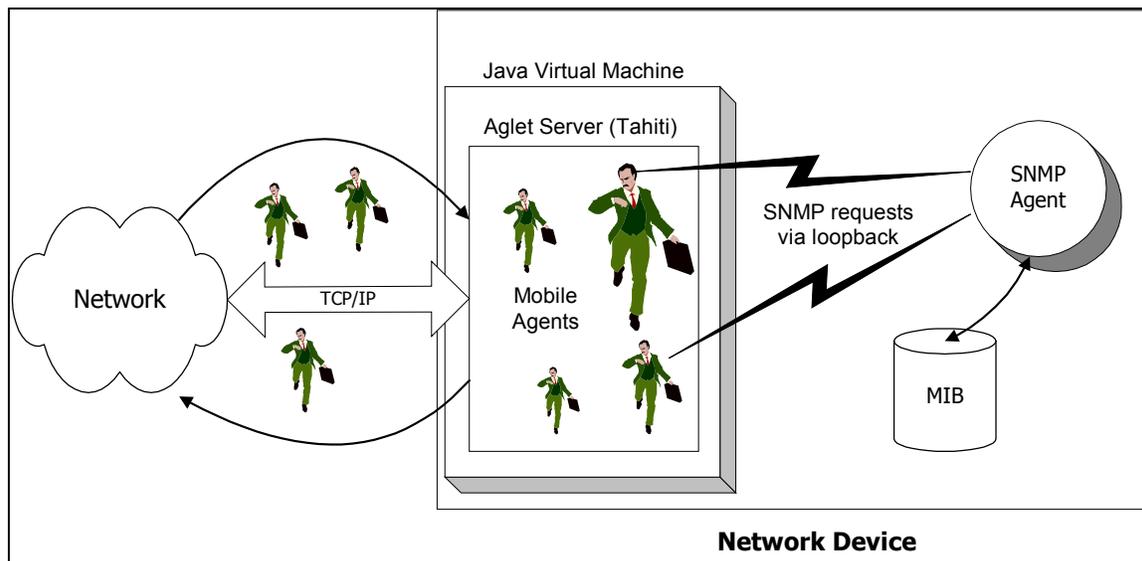
Figure 3: Network device side of implementation

Network devices have to provide a JVM in addition to the SNMP agent. Tahiti runs on every network device as the context server for incoming aglets. Arriving aglets are authenticated and restricted to security policies. They communicate with the SNMP agent via UDP packets, just like 'ordinary' SNMP-based management applications. The difference is that *no* actual network traffic is generated at all since these sockets are directed towards the 'loopback' device. Once an aglet has finished its job it will dispatch itself to the next destination on its itinerary.

3.2 Security of Aglets compared to SNMP

The fact that SNMP uses the unreliable, connectionless UDP rather than reliable, connection-oriented TCP reduces its security. An attacker can masquerade as a management station or a network device and send out malicious UDP packets to the well-known SNMP ports (161, 162) or corrupting ongoing SNMP request-response 'sessions'. Version 2 of SNMP did not provide

any substantial security enhancements[4] over version 1, other than proposing some mechanisms that were never included in the SNMPv2 standards.

Version 3 of SNMP incorporates a sophisticated and complex security model that provides for authentication and confidentiality, while allowing the user to specify the underlying cryptographic algorithms. The model is based on symmetric cryptography and its default implementation makes use of the DES and MD5 algorithms [28]. The cryptographic keys are automatically generated from a password. This contributes heavily to the solution of the key storage and management problems, but leaves the system vulnerable to dictionary and brute-force attacks, even though the password is concatenated, hashed, and `salted' with a network device's identity. As is well known, `crackers', which can also act as password testers, are generally available[5], including those that can perform a hash on similarly-encoded Linux passwords (though without concatenation). It is, therefore, up to the network managers to select a 'secure' password not accessible by an off-line attack, possibly a graphical password [38], and to change it frequently.

ATP, in contrast to normal usage for all versions of SNMP, uses the connection-oriented TCP with the port number able to be explicitly changed from the default by the network administrator. This has had the advantage that it has become possible to configure firewalls that block UDP traffic on SNMP ports, effectively preventing SNMP-based attacks. At the same time the TCP port used by the current aglet-based system can be left open. Another way of protecting the system against SNMP-based attacks is by disabling the capability of managing the devices remotely via SNMP. This was achieved by configuring the SNMP agent at network devices such that it only accepted SNMP PDUs (Protocol Data Units) from the "loopback" device, in effect only from the aglets that visit the device. The technique was verified for both Linux and Windows machines.

4 Security in the Aglet SNMP hybrid system

4.1 Authentication in Aglets

Two authentication schemes were considered: a signature-based; and one based on the ATP protocol. The signature-based authentication scheme would work as follows: Aglet bytecode is placed into JAR files that are then digitally signed with the private asymmetric key of the principal, a given NMS. The public keys of management stations are distributed and stored at

---

[4] Security was considered in a transitional version of SNMP, SNMPv2u.
[5] For example, a list is given on http://www.mycert.mimos.my/resource/cracker.htm#crack (current 17 July 2002).

every network device. The network device can then verify the signature of incoming mobile agent bytecode and set up security policies accordingly. For example, the policy of a network device may specify that only management stations A and B may 'manage' the device and that management station C can also perform additional, security-sensitive tasks (such as updating the key data base).

The main advantage of the signature-based authentication scheme is that it more naturally follows the architecture of mobile agent based network management: It is not necessary to authenticate every pair of communicating context servers since it is assumed that in a closed system like this servers may be trusted (network devices lie under the same administrative domain). Instead it is enough to be sure that the incoming aglet originates from a legitimate management station and that it has not been tampered with. Other advantages are

- This scheme does not generate additional network traffic at all. In fact, bytecode is 'forced' into JAR files which are compressed files ultimately bringing down network traffic to a minimum.
- It is also flexible because a given management station's key can be changed independently without affecting the functionality or security of other management stations. Some management stations may have more secure (longer) keys than others. Different management stations may choose to use different algorithms for their asymmetric keys.
- If a management station's private key is compromised, an attacker can only perform that management station's privileged actions. Thus it should be easy to isolate the problem and change the compromised key.
- The network layer address and DNS name of a management station can be changed without affecting the authentication mechanism (this may be unavoidable when the network topology changes).

The main disadvantage of this scheme is that it is vulnerable to replay attacks. However, as demonstrated in [23], Java policies may be based on digital signatures *and* code bases at the same time. This means that, if policies at network devices take into account *both* signatures *and* network layer addresses of management stations (encoded in the code base), a replay attack will only succeed if the attacker replays the captured aglet *while* spoofing the corresponding management station's IP address. If the DNS nameservice is used then the attacker must perform DNS masquerading instead of IP spoofing. This makes replay attacks more difficult (because of the need to impersonate a DNS server), although not impossible. One problem is that, in practice,

a management station's IP address may change and as a result not be taken into account at the network device security policies.

The current system provides an additional technique whereby replay attacks are effectively detected if they occur outside a certain time window. Ideally to do so requires a secure clock (Section 5.2.2). This will be effective even if an attacker is able to capture aglets, replay them at a later time and at the same time perform IP spoofing or DNS masquerading. Finally, the use of duplicated public keys can be a weakness, and this issue is taken up in Section 5.2.

Unfortunately, despite the advantage of code signing, it is not supported in the open source version of the Aglets Framework (version 2.0.2) at the time of writing. This meant that the only authentication scheme that can be supported by the open-source version of the current system is the one based on ATP. This scheme is weaker compared to code signing in that every server that participates in the domain is required to keep a common secret. It is expected that aglets will support code signing in the near future since it is a fundamental component of the standard Java security architecture.

Authentication under the ATP scheme is carried out based on a shared secret file. That file contains a secret symmetric key that is digitally signed and common for all Tahiti servers of the security domain. The shared secret is 'imported' every time Tahiti starts, after its signature is verified. It is stored in plaintext and must therefore be protected accordingly. The actual authentication is carried out by the ATP protocol every time an aglet is to be transferred between two Tahiti servers. From the aglets specification [39], it works as follows:

1. The sending Tahiti sends a nonce (a randomly generated integer value) to the receiving Tahiti server.
2. The receiving server combines the nonce with the shared secret and computes a secure hash value based on an algorithm like MD5 or SHA [28].
3. The secure hash value is sent back to the sending Tahiti together with another nonce, this time originating from the receiving Tahiti.
4. The sending Tahiti compares the hash value it received from the communicating party and compares it to its own version that it already has computed. If they match, the other end must be part of the security domain (since it must know the shared secret) and is therefore authenticated.
5. The sending Tahiti then combines the shared secret with the receiving Tahiti's nonce and sends back the secure hash value it computes.

6. The receiving Tahiti then repeats step (4) in effect authenticating the fact that it is receiving an aglet from a legitimate server.

7. Assuming successful authentication, the aglet is transferred.

The main advantage of this challenge-response scheme is that replay attacks are effectively prevented: Even if an attacker is able to capture an aglet while in transit the modified aglet will not be able to authenticate itself, as without knowledge of the shared secret it will not be able to compute the response to the randomly generated challenge. However, this authentication scheme has a number of drawbacks when compared to the signature-based one.

- There is no flexibility in the assignment of keys for management stations. Authentication is carried out for all servers based on the same shared secret. Security policies of network devices may however still be fine-tuned based on the code base (IP address or DNS name) of management stations.

- If the shared secret is compromised then the whole domain is exposed to dangers, as there will be no way to distinguish a legitimate server from an illegitimate one. (However, if code base directives are used in security policies an attacker must still be able to perform IP spoofing or DNS masquerading in order to provide authentication for malicious aglets).

- The scheme authenticates every aglet transfer between Tahiti servers. This may be unnecessary for this type of application.

- Additional traffic is generated for every aglet transfer. Given the fact that aglets are transferred via the connection-oriented TCP anyway (which needs connection establishment and tear-down), the extra three packets of the challenge-response of ATP may not impose a considerable overhead.

4.2 Securing the aglet's state.

An attacker may tamper with the aglet state, *i.e.* the data its carries while in transit, causing the aglets to behave differently as supposed, or causing the aglet to visit other hosts than those intended. Altering the aglet's state may also cause the aglet to report false information to the management application, thus confusing the system, triggering false alarms or hiding real ones. Furthermore, the state of the aglet must be protected against an eavesdropping attack as it will contain sensitive administrative information. It is clear that protecting the aglet's data state is almost as important as protecting the aglet's bytecode, yet the data state is neglected in some security systems.

In order to provide authentication and confidentiality of aglet's data state, a cryptographic model was developed. The model requires that management stations keep private/public key pairs, distributing the public keys to network devices. The keys may or may not be the same as the ones used for authentication of aglet bytecode. The current implementation uses a per-aglet key pair, where every aglet has its own private and public key in order to secure its state. The symmetric encryption algorithm chosen for this implementation is Rijndael, the Advanced Encryption Standard (AES) [40], and the asymmetric one is RSA. The digital signature is constructed by encrypting a MD5 hash value using a RSA private key [28]. As is well known, symmetric key encryption results in considerably reduced computation and, hence, symmetric keys are generally themselves encrypted for distribution via an asymmetric key.

The management station assigns an itinerary object to every outgoing aglet. This itinerary class, called `SecItinerary', was specifically developed for the hybrid SNMP system, and is protected against tampering through a digital signature using the aglet's private asymmetric key before it is sent out. The itinerary also contains a timestamp (in order to detect replay attacks) and some other variables that must be protected against tampering. The model is illustrated in Figure 4.
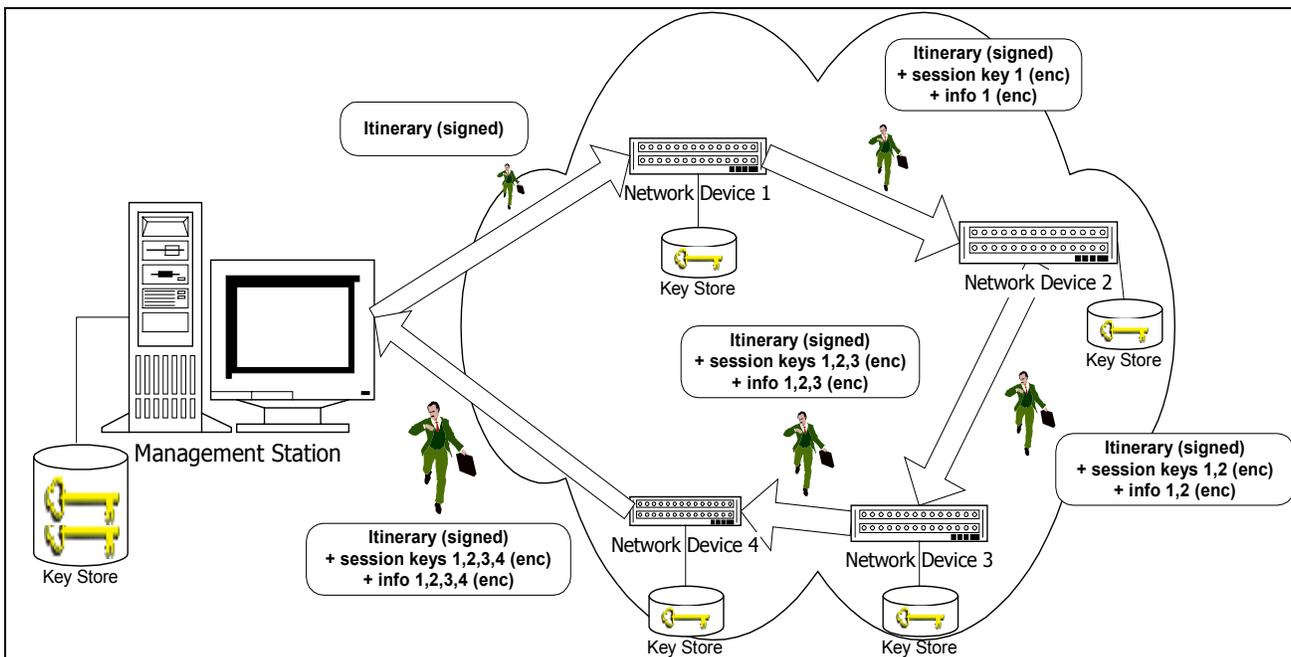
Figure 4: Security model of aglet state.

In reference to Figure 4, the exact sequence of events that take place is:

1. The management application decides that it has to send a mobile agent to a set of network devices, say 1, 2, 3 and 4, to perform a job. For example the aglet's name is `InfoCollect' and its job is to collect information from every network device it visits.

2. The management application constructs an itinerary for the InfoCollect aglet and digitally signs it with the aglet's private key that it keeps in its keystore. The aglet is now ready for its journey and is dispatched to its first destination.

3. After authentication of the aglet's bytecode and application of the appropriate policy, the first thing the aglet does is to verify the signature of its itinerary. This is possible because the public key of the InfoCollect aglet is kept in the key store of every network device.

4. Once the itinerary is verified, the aglet begins its job querying the local SNMP agent. It constructs its information objects that it will ultimately hand over to the management application when it will return.

5. The aglet then generates a random symmetric session key and encrypts (Rijndael) the information object(s) with it.

6. It encrypts the Rijndael key itself asymmetrically with its public key (the public key of the InfoCollect aglet).

7. The aglet then dispatches itself to its next destination (network device 2), taking with it the encrypted objects generated at network device 1.

8. At every network device the aglet visits the same sequence of events is repeated (3 – 7), until it reaches the end of its itinerary. At that point the aglet will have accumulated all encrypted session key and information objects from all the network devices of its itinerary.

9. The aglet will then dispatch itself back to the management station where it then hands over all encrypted objects (session keys and information objects) to the management application. This concludes the aglet's life cycle and it may now be disposed of.

10. The management application decrypts the session keys using the InfoCollect aglet's private key and the corresponding information objects using the revealed session keys.

This encryption scheme, when combined with one of the authentication effectively prevents eavesdropping and tampering attacks. It is worth noting that the aglet itself never has access to its private key. Private keys are stored in an encrypted form at the management station's key database and are protected by a password, which does not need to be permanently stored by the management application, but might be held on a smart card.

The implementation of this scheme, however, must also take into account the fact that network traffic must be kept to a minimum and that other kind of attacks may be launched against the aglet framework. Such attacks could include aglets that try to specify their own version of security-sensitive classes, in effect compromising the system's security. This point is returned to in Section 5.1.

4.4 Security providers

Authentication of byte code is a matter of careful configuration and specification of security policies and distribution of keys. As only bytecode is subject to authentication (in the signature-based approach), there is no assurance that the state of aglets has not been tampered with by an attacker while in transit. Furthermore, there are no existing mechanisms of encryption in aglets that may be used in order to provide confidentiality of the information gathered at remote network devices.

Therefore aglets framework was extended in order to provide encryption and authentication services on the level of an aglet's state. The Java Cryptography Extension (JCE) is a common API programmers rely on for encryption services. Implementation of the encryption algorithms (at the lowest level) relies on security providers. Of the security providers considered: one did not implement the RSA algorithm; another was not fully compatible with the JCE API; and two others were commercial products. Finally, the security provider of the Institute for Applied Information processing and Communications (IAIK) [41] was chosen. Care was taken to make it possible to plug any Security Provider into the system, without much programmatic effort. The provider, as well as the cryptographic algorithms used for digital signing and encryption, can be specified in the "aglets properties" configuration file.

5 Implementation considerations

This auxiliary part of the hybrid system is of more recent origin and therefore less consolidated. In particular, inter-aglet messages are employed, which, though encrypted and authenticated, may be an additional target. However, the extent of messaging in the key distribution aglets of Section 5.2 is strictly limited. Messages are only sent between management application and aglets, or between stationary master and slave.

5.1 Local security class package

It is important to be able to provide the infrastructure for secure and thin (small in code size) aglets. Hence the security-relevant code must as much as possible be part of the aglets

framework itself and therefore be available locally at every network device. This has another important advantage: as the Tahiti Class Loader first tries to load classes that are locally available, it is not possible for an incoming aglet to overload security-relevant code (provided the classes are declared 'final'). Therefore, the aglets framework was extended by a new Java package, the `agletsec` package[6]. This package allows aglets to encrypt, decrypt and digitally sign data at the object level. The aglet needs a few lines of code to make use of the services, thus keeping its size (and therefore network traffic) to a minimum.

5.2 Key distribution and clock synchronization

Every aglet has a (public, private) key pair in the keystore of the management station and a copy of its public key in the keystore of every managed network device. It is of great importance to distribute the public keys of aglets securely to every network device. Although an attacker who is in the possession of an aglet's public key cannot perform any eavesdropping based only on that public key, it is important to make sure that an aglet's public key is genuine and not maliciously altered by an attacker. A maliciously altered public key could authenticate an attacker's aglet's secure itinerary.

A weakness that *may* be open to exploitation is the use of duplicate keys. Key security has been highlighted as a problem in secure file systems for PCs. For example, there is the risk of keys being paged out to disc, and subsequently being accessible [42].

Similarly, it is important to keep clocks of network devices and management stations in synchrony in order to be able to detect replay and delay or `Cinderella' attacks. The Network Time Protocol (NTP) [43] achieves precise and secure clock synchronization, with clock voting [44]. In order, however, to avoid the increased complexity of an additional protocol in the system as well as to add flexibility, a mobile agent clock synchronization scheme was also developed.

5.2.1 Mobile agent-based key distribution

An MA-based key update mechanism was developed and implemented that only requires one key pair at a management station and public key copies of only that one key pair at every network device. That asymmetric key pair goes under the alias "KeyMaster". It is crucial that public "KeyMaster" keys are distributed securely to network devices. A text-oriented tool that extracts the public keys of a keystore file and saves them in another keystore file was developed

---

[6] Open source access to this code is available at http://www.essex.ac.uk/ese/research/mma_lab/index.htm.

for this purpose. Public keys can now be distributed on an out-of-band channel to network devices, for example, on a floppy disk. The approach avoids the need for certification authorities (CA) that otherwise would be used to certify the fact that a given public key actually belongs to a given principal.

The model is based on the master-slave pattern and the protocol according to which public keys are updated involves the management application, a `KeyMaster` aglet and a `KeySlave` aglet. The model can be summarized as follows:

1. There is a need to update (create, renew, delete) the key pair for a new principal, say the aglet "someAglet".
2. The management application that spawns someAglet creates the KeyMaster aglet and assigns it a SecItinerary (as usual). This itinerary should contain all destinations for which the key update is desired.
3. The KeyMaster is a stationary aglet and first creates a new asymmetric key pair and a self-signed certificate containing the public key. These are the new keys to be used by someAglet.
4. The KeyMaster creates and spawns the KeySlave aglet and assigns it the same SecItinerary.
5. The KeySlave dispatches itself to its first destination.
6. The KeySlave generates a random symmetric session key and encrypts it with the KeyMaster's public key (that is kept locally at every network device).
7. The KeySlave sends the encrypted session key back to the KeyMaster as a message.
8. The KeyMaster decrypts the session key (using the KeyMaster private key) and encrypts the certificate (from step 3) *and* the alias 'someAglet' with the session key.
9. The KeyMaster sends the encrypted certificate *and* alias to the KeySlave as a message.
10. The KeySlave decrypts the certificate and alias and updates the local keystore accordingly. This will only succeed if the KeySlave was previously authenticated and permission to write to the key store is granted in the local policy file.
11. The encrypted session key (from step 6) is stored in the aglet's state together with the result of the whole operation (whether the update was successful or which error occurred), which is encrypted with the session key.
12. The KeySlave dispatches itself to its next destination and steps 6-11 are repeated until the KeySlave has traveled through its entire SecItinerary.

13. Once the KeySlave is back at the management station it "reports" its progress to the KeyMaster by means of a message that contains the accumulated session keys and results of every destination (the information collected from step 11).

14. The KeySlave disposes of itself since its lifecycle is over and the KeyMaster forwards the "report" to the management application, which in turn can now decrypt the information using the KeyMaster's private key and the revealed session keys.

15. The KeyMaster updates the management station's keystore with the new key pair.

16. The KeyMaster disposes of itself since its lifecycle is over.

It is worth noting that more than one key can be updated at once and that the KeyMaster keys themselves can also be updated using the above protocol. The master-slave mechanism is transparent to the management application.

5.2.2 Mobile agent based clock synchronization

It is not possible to update system clocks in Java, because this is a platform-specific operation and sometimes involves security checks imposed by the operating system. Clock synchronization can nevertheless be achieved by maintaining a local *offset* variable at each server that indicates by how much the local clock is slow or fast, compared to a 'master' clock. One management station is designated to provide the 'master' clock, and send out aglets that 'synchronize' the local clocks of the other servers in the domain. This management station could synchronize its clock using NTP, so that a global time mechanism is achieved, or some other clock source could be used removing NTP dependence. Another possibility is to chose a random start at the management station, or use a logical clock. It also proves important *not* to grant permission to change the local clock offset, at the master clock host, in order to avoid 'accidental' updates of the master clock. The synchronization protocol is just another application of the mobile agent based key distribution protocol, as defined in Section 5.2.1, involving a TimeMaster and TimeSlave aglet.

5.3 Logging

The open source Log4J [45] provides an API that the aglet programmer may use in order to log any kind of information (debug, error, exceptions, security-relevant, fatal errors) to virtually any kind of medium. This logging scheme has been particularly important in this distributed system, as it can be used to detect attacks that have been attempted (whether

successfully or not) on the system, including those when the manager was temporarily absent from the management station.

6 Example MAs

The example aglets make use of the AdventNet SNMP API[7] in order to communicate with the SNMP agents of the network devices. The SNMP API is installed locally at every server. Protocol knowledge and code is thereby available locally at the devices with the result that the size of aglets can be dramatically reduced. In fact, the SecAglet (Section 6.1) was less than 3Kbytes (in a JAR file), which translates into only two IP datagrams over a standard Ethernet LAN.

6.1 SecAglet

The purpose of this aglet was to demonstrate that management information can be securely collected from network devices, via their SNMP agents. The aglet follows its SecItinerary (that was assigned to it by the management application), verifies its signature and the fact that it is still valid at every network device it migrates to. A new session key is generated at each server with which the management data is encrypted, while the session key itself is encrypted with the management station's public key. It should be noted here that, since encryption is performed on the object level, objects are able to be arbitrarily complex, as long as they are serializable.

6.2 TrapAglet

It is certainly not sufficient to provide only for that basic travel pattern of SecAglet, where a specific management task is to be performed serially at a number of network devices. Polling and health functions require continuous contact with the SNMP agent and are therefore perfectly suited for mobile agents, which can perform these tasks locally. Catching SNMP traps locally and taking appropriate action without having to bother the management station for simple or common situations can also be taken care of by mobile agents. These tasks require the mobile agent to remain resident at the device. The mobile agent should, however, still be able to communicate accumulated information (rather than raw data) to the management application, so that action can be taken about extraordinary events. The weakness of SNMPv1, in that there was no acknowledgement of critical events, can also be avoided.

---

[7] Downloadable from http://www.adventnet.com/products/snmp/index.html (current 16 July 2002).

The purpose of the TrapAglet was to demonstrate that it is possible to follow such a travel pattern. Like the SecAglet, the TrapAglet visits a series of network devices and returns back to the management station in order to 'report'. The difference is that, at each destination, TrapAglet clones itself. This process can be thought of as the aglet installing itself on each network device. The clones listen locally for SNMP traps. If information resulting from a trap has to be communicated back to the management application, the scheme is similar to that of key distribution. The information object(s) are encrypted using a randomly generated symmetric session key while the key itself is encrypted with the asymmetric public key of the TrapAglet, and everything is sent as a message. If need be, a reply to that message (possibly encrypted under the same session key) gives the aglet information on further actions it should take.

7 Conclusions

Secure network management has a vital role to play in providing network survivability. This paper has demonstrated that it is now possible to construct a secure but de-centralized management system using standard components, namely Java aglets. All the classes developed are open source and available, allowing a secure solution to evolve through discussion amongst domain experts. To this end, the paper has presented reasonable detail, not a discussion of development class features, but not also only a system design level view. Two methods of authentication have been presented with a proposed code signing method preferred. The data state is cryptographically protected. Relevant classes are stored locally to reduce the aglet size. Key distribution by aglet has been prototyped. Some care has been taken to prevent active attacks as the aglets undertake an itinerary itself protected. However, for the reasons stated, it is not expected that this solution will be the final one. Another issue that has been addressed, and which is often taken for granted, is availability of a secure clock. The paper has included discussion of the correct role for mobile agents in network management. Evidently, aglets may have some advantages over the security model now standardized in SNMPv3. Equally some tasks appear more suitable for a centralized approach, given that a mobile agent can also introduce response latency. Traps are a particular problem as they introduce dynamic communication, which may give an opportunity for an attacker to disrupt the NMS. However, traps also have a security in role in logging error conditions, which may indicate a breach. Thus, a trap aglet has been developed to assess how an aglet might be delegated to this role. Future work involves extensive testing of this aglet system beyond the prototype phase.

References

1. Subramanian, M., Network Management, Addison-Wesley, Reading, MA, 2000.
2. W. Stallings, SNMP, SNMPv2, SNMPv3 and RMON 1 and 2, 3rd ed., Addison-Wesley, Reading, MA, 1999.
3. Hewlett Packard, Openview User's Guide, 1992.
4. SunSoft, SunNet Manager Reference Manual, 1994.
5. Information Technology, Open Systems Interconnection, Systems Management Overview, ISO/IEC, Sept., 1991.
6. Case, J., Fedor, M., Scoffstall, M., Davin, J., A Simple Network Managament Protocol (SNMP), RFC 1451, 1993.
7. CERT Advisory CA-2002-03: Multiple Vulnerabilities in Many Implementations of the Simple Network Protocol (SNMP), 2002, http://www.cert.org/advisories/CA-2002-03.html (current 11 March 2002).
8. PROTOS Test-Suite: c06-snmpv1, Oulu University Secure Programming Group, 2002, http://www.ee.oulu.fi/reseach/ouspg/protos (current 11 March 2002).
9. Ziegler, R. L., Linux Firewalls, New Riders, Indianapolis, IND, 2000.
10. Baldi, M., Gai, S., and Picco, G.P., Exploiting Code Mobility in Decentralised and Flexible Network Management, Proceedings of the 1st Workshop on Mobile Agents (MA'97), LNCS vol. 1219, pp. 13-26, 1997.
11. Sprenkels, R., and Martin-Flatin, J. P., Bulk Transfers of MIB Data, The Simple Times, vol. 7, no. 1, pp. 1-7, 1999.
12. Levi, D. and Schoenwaelder, J., Definitions of Managed Objects for the Delegation of Management Scripts, RFC 2592, 1999.
13. Mazumdar, S., Inter-domain Management between CORBA and SNMP: Web-based Management – CORBA/SNMP Gateway Approach, Proceedings of the 7th IFIP/IEEE International Workshop on Distributed Systems, Operations & Research (DSOM'96), 1996.
14. Wellens, C. and Auerbach, K., Towards Useful Management, Simple Times, vol. 4, no. 3, pp. 1-6, 1996.
15. Strassner, J. and Baker, F., Directory Enabled Networking, Macmillan Technical Publishing, 1999.
16. Meyer, K., Erzlinger, M., Betser, J., Sunshine, C., Goldschmidt, G., and Yemini, Y., Decentralizing Control and Intelligence in Network Management, Proceedings of the 4th International Symposium on Integrated Network Management (ISINM'95), pp. 4-16, 1995.
17. Liotta, A., Knight, G., and Pavlou G., Modelling Network and System Monitoring over the Internet with Mobile Agents, Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS'98), pp. 303-312, 1998.
18. Papavassiliou, S. Puliafito, A., and Ye, J., Mobile Agent-based Approach for Efficient Network Management and Resource Allocation: Framework and Applications, IEEE Journal on Selected Areas in Communications, vol. 20, no. 4, pp. 858-872, 2002.
19. Kerckhoffs, A., La Cryptographie Militaire, Journal des Sciences Militaires, vol. IX, pp. 5-38, 1883.
20. Stubblefield, A., Ioannadis, J., and Rubin, A. D., Using the Fluhrer, Mantin, and Shamir Attack to Break WEP, Technical Report TD-4zCPZZ, AT&T Labs., 2001.
21. Oaks, S., Java Security, 2nd ed., O'Reilly, Beijing, 2001.
22. Volpanao, D. and Smith, D., language Issues in Mobile Program Security, In Mobile Agents and security, ed. Vigna, G., LNCS 1419, pp. 25-43, 1998.
23. Lange, D.B. and Oshima, M., Programming and Deploying Java Mobile Agents with Aglets, Addison-Wesley, Reading, MA, 1998, source code available at http://www.sourceforge.org/projects/aglets (current 15 July, 2002)
24. L. Silva, P. Simões, J. Gabriel e Silva, J. Boavida, P. Monteiro, J. Rebhan, C. Renato, L. Almeida, N. Stohr, Using Mobile Agents for the Management of Telecommunication

Networks, Proceedings of ConfTele99 – The 2nd Conference on Telecommunications, Instituto de Telecomunicações, Sesimbra, Portugal, April 1999.

25. B. Paguek, Y. Wang, T. White, Integration of Mobile Agents with SNMP: Why and How, In IEEE Network Operations and Management Symposium, 2000.
26. M. Zapf, K. Hermann, K. Geihs, Decentralized SNMP Management with Mobile Agents, In Proceedings of IM '99, Boston, May 1999.
27. Bellavista, P. Corradi, A., and Stefanelli, C., An Open Secure Mobile Agent Framework for Systems Management, Journal of Network and Systems Management, vol. 7, no. 3, 1999.
28. Schneier, B., Applied Cryptography, 2nd ed., Wiley, New York, 1996.
29. Carzaniga, A., Picco, G. P., and Vigna, G., Designing Distributed Applications with Mobile Code Paradigms, Proceedings of the 19th International Conference on Software Engineering, 1997.
30. Walsh, T., Paciorek, N., and Wong, D., Security and Reliability in Concordia^{tm}, Proceedings of the 31st Hawaii International Conference on Systems Sciences, vol. VII, pp.44-53, 1998.
31. Lavian, T., Jaeger, R. F., and Hollingsworth, J. K., Open Programmable Architecture for Java-enabled Network Devices, Hot Interconnects, pp. 265-277, 1999.
32. Anderson, R., *Security Engineering*, Wiley, New York, 2001.
33. Hohl, F. Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts, In Mobile Agent Security, Vigna, G. (ed.), pp. 92-113, LNCS 1419, 1998.
34. Karnik, N. H. and Tripathi, A. R., Security in the Ajanta Mobile Agent System, Software: Practice and Experience, vol. 3, no. 4, pp. 301-329, 1999.
35. Gray, R.S., Kotz, D., Cybenko, G., Rus, D., D'Agents: Security in a Multiple-language, Mobile-agent System, in Mobile Agents and Security, Vigna, G. (ed.), pp. 188-205, LNCS 1419, 1998.
36. Garfinkel, S., *PGP*, O'Reilly, Cambridge, 1995.
37. Karjoth, G., Lange, D. B., and Oshima, M., A Security Model for Aglets, in Mobile Agents and Security, Vigna, G. (ed.), pp. 154-187, LNCS 1419, 1998.
38. Jermyn, I., Mayer, A., Monrose, F., Reiter, M.K., and Rubin, A.D., The Design and Analysis of Graphical Passwords, In Proceedings of the 8th USENIX Security Symposium, pp. 1-14, 1999.
39. Oshima, M., and Karjuth, G., Aglets Specification, IBM, Japan, 1997.
40. Nechvatel, J., Bassham, L., Burr, W., Dworkin, M., Foti, J., and Roback, E., Report on the Development of the Advanced Encryption Standard, NIST, 2000.
41. Javadoc for IAIK-JCE 3.01, IAIK at Graz University of Technology, http://jcewww.iaik.tu-graz.ac.at/products/01_jce/documentation/index.php (current 22nd June 2002)
42. Encrypting File System for Windows 2000 White Paper, Microsoft Corp., 1998.
43. Mills, D.L., Internet time synchronization: The Network Time Protocol, Network Working Group Request for Comments: 1129, pp. 1-29, 1989.
44. Mills, D.L. Public key cryptography for the Network Time Protocol. Electrical Engineering Report 00-5-1, pp. 1-23, University of Delaware, 2000.
45. Gulcu, C., Log4j Delivers Control over Logging, JavaWorld, Nov. 2000.