A multi-purpose authorization and policy management agent

Camillo Särs

Supervisor: Professor Arto Karila
Instructor: Ari Hyppönen
The management and administration of computer networks has become increasingly difficult due to the increased use of the Internet. New security threats emerge when previously closed systems are connected, either directly or through the Internet, to form large heterogeneous networks. Legacy solutions neither provide adequate protection against the new threats, nor do they scale well enough to be relied on in ever larger networks. New solutions need to be able to dynamically adapt to their environment and to always react in a predictable and secure manner. Our solution is to introduce an intelligent authorization and policy management agent on every network node.

The requirements for an intelligent agent are diverse and partially contradict each other. We introduce a model where the agent works at a high level of abstraction and provides client applications with several open APIs. Authorization certificates provide us with a powerful means of expressing system policies and user authorizations. Authorizations can be assigned to individual principals who may or may not have the right to further delegate the assigned authority.

A reference monitor at the core of the agent enforces the system policy and unconditionally rejects unauthorized operations. Authorized applications and users have access to a range of security-critical operations including software management, auditing, alerting, and user-authentication services. The services provided by the agent can be extended by third parties using the provided SPIs.

**Keywords:** authorization, certificate, delegation, management, security policy, SPKI
Tietokoneverkkojen hallinta ja ylläpito on tullut yhä vaikeammaksi Internetin käytön kasvun myötä. Uusia turvauhkia ilmestyy, kun aiemmin suljettuja järjestelmiä yhdistetään joko suoraan tai Internetin välityksellä muodostaen suuria heterogeenisia verkkoja. Vanhat ratkaisut eivät tarjoa riittävää suojaaa uusilta uhilta, eivätkä ne myöskään skaalaudu riittävän luotettavasti yhäs isommissa verkoissa. Uusien ratkaisujen on kyettävä dynaamisesti sopeutumaan ympäristöönsä ja niiden on aina reagoitava ennakoitavasti ja luotettavasti muutoksiin. Ratkaisu on tuoda jokaisen verkkosolmuun älykäs agentti joka hoitaa valtuuksien ja turvakäytännön hallinnan.


Referenssimonitori agentin ytimessä valvoo järjestelmän turvakäytännön toteuttamista ja hylkää ehdottomasti toiminnot, joihin ei ole valtuuksia. Valtuutetut sovellukset ja käyttäjät voivat käyttää laajaa joukkoa turvakriittisiä toimintoja mukaan lukien ohjelmistohallinnan, auditoinnin, hälytykset ja käyttäjän autentikointipalvelut. Kolmannet osapuolet voivat laajentaa agentin tarjoamia palveluja.

<table>
<thead>
<tr>
<th>Tekijä:</th>
<th>Camillo Särs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Työn nimi:</td>
<td>Monikäyttöinen valtuutus- ja turvakäytännönhallinta-agentti</td>
</tr>
<tr>
<td>Päivämäärä:</td>
<td>1998-04-03</td>
</tr>
<tr>
<td>Sivumäärä:</td>
<td>73</td>
</tr>
<tr>
<td>Osasto:</td>
<td>Tietotekniikan osasto</td>
</tr>
<tr>
<td>Professuuri:</td>
<td>Tik-110 Tietokoneverkot</td>
</tr>
<tr>
<td>Työn valvoja:</td>
<td>Professori Arto Karila</td>
</tr>
<tr>
<td>Työn ohjaaja:</td>
<td>Ari Hyppönen</td>
</tr>
</tbody>
</table>

Hakusanat: delegointi, hallinta, turvakäytäntö, sertifikaatti, SPKI, valtuutus
Preface

This thesis is a result of the good cooperation between the Telecommunication Software and Multimedia Laboratory at Helsinki University of Technology and Data Fellows Ltd. Without the excellent tuition in computer security I have enjoyed at HUT and the large amount of freedom I have enjoyed at Data Fellows, I would not have had the opportunity to write this thesis.

I want to thank my professor, Arto Karila, for his support and patience. He never lost faith in me and supported me throughout the process.

Pekka Nikander deserves my warmest gratitude as his conviction and knowledge of authorization certificates and certificate loops have helped me a lot.

I have learned a lot from Carl Ellison, the author of SPKI, and from Ed Gerck, who taught me to distinguish between authorization and trust.

A number of Fellows have helped me in my work, and much of their work is reflected in mine. I hope that this thesis in turn will influence their work and help us work together to make the agent a reality. My instructor Ari Hyppönen has helped me define and outline the area of the work.

Finally, I must thank my fiancée Jonna for having the patience to listen to me and my ideas, no matter how odd they may have sounded at the time.

Otaniemi 1998-04-03

Camillo Särs
# Table of Contents

Abstract .......................................................................................................................... ii
Tiivistelmä ....................................................................................................................... iii
Preface ............................................................................................................................ iv
Table of Contents .......................................................................................................... v

Terms .............................................................................................................................. vi

1. Introduction ............................................................................................................... 1
   1.1 Background ......................................................................................................... 1
   1.2 Objective .......................................................................................................... 1
   1.3 Organization .................................................................................................... 2

2. F-Agent .................................................................................................................... 3
   2.1 Description ....................................................................................................... 3
   2.2 Scheduled execution ....................................................................................... 4
   2.3 Task distribution ........................................................................................... 5
   2.4 Software updates ......................................................................................... 6
   2.5 Alerting and bulletins .................................................................................. 6
   2.6 Limitations .................................................................................................... 7

3. Base requirements .................................................................................................. 8
   3.1 Perspective ...................................................................................................... 8
   3.2 Functions ....................................................................................................... 11
   3.3 User roles ..................................................................................................... 12
   3.4 General constraints ..................................................................................... 13
   3.5 Assumptions and dependencies ..................................................................... 13

4. Main Concepts ....................................................................................................... 15
   4.1 Identity and names ......................................................................................... 15
   4.2 Authentication vs. authorization .................................................................... 18
   4.3 Trust ............................................................................................................... 20
   4.4 Delegating authority .................................................................................... 21
   4.5 Policy ............................................................................................................. 23
   4.6 Cryptography ............................................................................................... 24

5. Structure .................................................................................................................. 30
   5.1 Framework & technology ............................................................................ 30
   5.2 Policy-based management .......................................................................... 34
   5.3 Authorization agent ..................................................................................... 37
   5.4 Software updates ......................................................................................... 41
   5.5 Configuration management ........................................................................ 43
   5.6 Auditing .......................................................................................................... 45
   5.7 Alerting .......................................................................................................... 48
   5.8 Hardware abstraction .................................................................................. 51
   5.9 Public key infrastructure ............................................................................ 53

6. Conclusions ............................................................................................................. 57
   6.1 Suggestions for further study ......................................................................... 58

7. Summary .................................................................................................................. 60

References .................................................................................................................... 62

Appendix A: Open APIs ............................................................................................. 65
<table>
<thead>
<tr>
<th>Terms</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface. A collection of functions that applications can use to access a service.</td>
</tr>
<tr>
<td>auditing</td>
<td>The automatic recording of events on a computer system in a secure computerized log.</td>
</tr>
<tr>
<td>authentication</td>
<td>The act of proving one's identity.</td>
</tr>
<tr>
<td>authorization</td>
<td>The right to perform an action on an object. Also the act of proving this right.</td>
</tr>
<tr>
<td>authorization certificate</td>
<td>A certificate binding an authorization to a principal.</td>
</tr>
<tr>
<td>capability</td>
<td>Possessing a capability grants the holder some authorization, independent of the holder's identity.</td>
</tr>
<tr>
<td>certificate</td>
<td>A digitally signed record that binds some characteristic to a principal or set of principals.</td>
</tr>
<tr>
<td>certificate chain</td>
<td>A set of related authorization certificates with delegations that can be used to deduce an authorization.</td>
</tr>
<tr>
<td>delegation</td>
<td>The transfer of authority from one principal to another.</td>
</tr>
<tr>
<td>FAR</td>
<td>False Acceptance Rate. Indicates how probable an incorrect identification by a (biometric) device is.</td>
</tr>
<tr>
<td>global name</td>
<td>A name in the namespace of the global community. The only global names we recognize are public keys, which identify their principals.</td>
</tr>
<tr>
<td>HAL</td>
<td>Hardware Abstraction Layer. A software layer that hides the intricate details of various hardware devices.</td>
</tr>
<tr>
<td>identity certificate</td>
<td>A certificate binding a name to a principal.</td>
</tr>
<tr>
<td>identity</td>
<td>The least amount of information required to unambiguously associate a principal with a given characteristic. Usually, this is</td>
</tr>
</tbody>
</table>
simply “key”. This definition is valid in the scope of a single authorization and loses its meaning when applied to a broader context.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPsec</td>
<td>Internet Protocol Security. A security protocol that enhances IP by adding confidentiality and integrity services.</td>
</tr>
<tr>
<td>kernel mode</td>
<td>The part of an operating system core where code is executed at a high privilege level and with few operating restrictions. Ordinary applications cannot run in kernel mode.</td>
</tr>
<tr>
<td>key</td>
<td>A public key identifying a principal. When we refer to “key” we typically mean “the principal identified by this public key”. The reason for this indirection is self-evident, the private key is not public knowledge.</td>
</tr>
<tr>
<td>keyholder</td>
<td>A person or entity in possession of a certain principal. This entity is seldom directly involved. Instead the principal “speaks for” the entity.</td>
</tr>
<tr>
<td>name</td>
<td>A token uniquely identifying a principal within a certain namespace, also known as a domain of interpretation. The definition of a name may refer explicitly to names in other namespaces.</td>
</tr>
<tr>
<td>namespace</td>
<td>A set of names as defined by a principal.</td>
</tr>
<tr>
<td>PGP</td>
<td>Pretty Good Privacy. An application for public-key encryption and key certification using a “web of trust”.</td>
</tr>
<tr>
<td>policy</td>
<td>The set of conditions under which users of a system can access the system’s resources.</td>
</tr>
<tr>
<td>policy-based management</td>
<td>Controlling the actions and configurations of a system using policy statements.</td>
</tr>
<tr>
<td><strong>principal</strong></td>
<td>A private (signature) key with a corresponding public key. The private key is the only entity that is capable of &quot;speech&quot; and is identified by its public key.</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>prover</strong></td>
<td>The keyholder that wishes to perform some action or obtain some authorization.</td>
</tr>
<tr>
<td><strong>reference monitor</strong></td>
<td>A conceptual model of a monitor that receives actions and either allows or denies their execution.</td>
</tr>
<tr>
<td><strong>role</strong></td>
<td>A collection of authorizations that can be assigned to a user.</td>
</tr>
<tr>
<td><strong>service (application)</strong></td>
<td>An application that is running on a host regardless of who is logged in, and which provides some service to other applications.</td>
</tr>
<tr>
<td><strong>SPI</strong></td>
<td>Service Provider Interface. A collection of functions used by applications and drivers to extend the functionality of a service.</td>
</tr>
<tr>
<td><strong>SPKI</strong></td>
<td>Simple Public Key Infrastructure. An authorization certificate framework, described in detail in [SPKIa-c].</td>
</tr>
<tr>
<td><strong>TCB</strong></td>
<td>Trusted Computing Base. The set of functions that must be trusted for a secure system to be able to operate securely.</td>
</tr>
<tr>
<td><strong>user mode</strong></td>
<td>The protected part of an operating system where user applications run and call kernel mode functions to access operating system services.</td>
</tr>
<tr>
<td><strong>verifier</strong></td>
<td>The keyholder that needs to verify the prover's request for action or authorization.</td>
</tr>
<tr>
<td><strong>X.509</strong></td>
<td>An identity certificate framework, described in detail in [X.509]</td>
</tr>
</tbody>
</table>
1. Introduction

1.1 Background
Data Fellows, Ltd., is a software house that specializes in computer security software. Its F-Secure product family consists of both anti-virus and cryptographic software which until recently were separate product families. We now aim to integrate the products more closely to gain both efficiency and security.

The need for this thesis grew out of several software development projects. The product family needed a clear roadmap that integrated many of the existing parts and provided a framework for new services. While building this roadmap, we noticed that many of the functions were common to the parts and should preferably be integrated into a general module.

Prior development history and customer feedback provided us with an excellent base for product development. We hope to focus on features that will bring added value in the long term rather than involve us in short-term development. We will also explore a few new technologies that seem to have a bright future.

The need for a centralized, intelligent management component is evident. We hope that this thesis will serve as a good foundation for future work.

1.2 Objective
We aim to define a set of requirements and combine them to form the frame of a requirements specification. Based on these requirements, we will present an initial model of a possible solution. The model will be based on our existing products, but will also incorporate features that will not be implemented for the time being.

The solution will allow both anti-virus and cryptographic software to share critical components. The benefits of this are several, including reducing the critical code base and avoiding duplication of functionality in the different products. It will also mean that a module that is added to one of the products can be used in other products as well, with little or no coding effort.

It is important to note that we do not aim for comprehensive coverage of all possible models; the area is far too broad for that. We will focus on one security model, based on
capabilities and authorization certificates, and its implications for product development.
Security is a core objective and we will integrate it into the central parts of our model.

Although the goal of this thesis is to provide concrete added value to the product development process, we recognize that it will only serve as a foundation for the actual development effort. As always, this will inevitably lead to changes and new innovations.

Some of the components we describe exist today; some resemble components that are being planned. This thesis is the merger of work done by others, and ideas and visions that we intend to integrate in the future. We cannot take credit for all the thoughts presented here, but have done our best to avoid presenting work by others as though it were our own.

1.3 Organization

This thesis is organized into seven sections.

Section 2 describes the current product and some of its shortcomings. It presents details from the product that can be re-used in part in the new agent.

Section 3 presents a few base requirements for further development. These requirements form the basis of the work. Other requirements are presented later in the work.

Section 4 introduces the main concepts and most of the references, and builds a base for the rest of the thesis. The concepts are important for understanding the philosophy of the agent, even though some of the concepts are not directly used in the thesis.

Section 5 describes the different features we will integrate into the agent and why they are needed. The space is too limited for a thorough presentation of each feature, but the key issues are presented and tied together to form the structure of the agent.

Section 6 presents our conclusions and suggestions for further work.

Section 7 summarizes the main issues in the thesis.
2. F-Agent

2.1 Description

F-Agent is an application that is part of the F-Secure Anti-Virus for Windows products. It is designed to provide the different anti-virus products with a number of critical services and to help administrators control the updating and configuration of anti-virus scanners in a network. Regular reliable updates are a critical part of an effective anti-virus protection framework. Running outdated scanners can be worse than having no protection at all, as it gives users a false sense of security.

F-Agent is designed to be started at system startup and mostly runs silently in the background. On network servers it is typically run as a service, which means that it is active even when nobody is logged in. On workstations, the ability to run as a service is less important, especially when you consider that installing a service requires administrator privileges. This is why many sites choose to run F-Agent as an application on end-user workstations.

When run as a service, F-Agent has access only to a restricted set of functions that depends on what account it is running under. To circumvent this, the application version is typically launched when a user logs in to provide the user interface and missing functionality.

F-Secure Anti-Virus and F-Agent communicate over a network using a shared communications directory on some file server. Updates, tasks, bulletins, messages, and "virus found" reports are all communicated by copying files to and from different subdirectories of the shared directory. This design is independent of the network protocol used and has proven to be very portable. Binding the mechanism to the network used would require several different versions of the products.

The tasks of the application-mode F-Agent as listed in the specification are:

- Run local scheduled tasks and poll for new tasks in the communications directory.
- Poll for updates in the communications directory.
• Post “virus found” messages, coming from the real-time scanner, in the communications directory.

• Run idle-time tasks after a preset period of idle-time.

• Provide a user interface and provide child processes with the required privileges to show their user interface.

The tasks of the service-mode F-Agent depend on the account it is run under and if the application-mode agent is running.

Under a “Local System” account:

• Poll and run local scheduled tasks.

• Provide the user interface.

• Run idle-time tasks after a preset period of idle-time.

• F-Agent cannot access a remote communications directory and therefore cannot notice updates, new tasks or post “virus found” messages.

On a user account, which is the recommended way of running the service:

• Poll and run local and remote scheduled tasks.

• Poll for updates in the communications directory.

• Post “virus found” messages from the real-time scanner.

• The account needs read and write access to the communication directory.

• F-Agent does not have the necessary privileges for a user interface and cannot notice user idle time.

2.2 Scheduled execution

F-Secure Anti-Virus allows the user or the administrator to schedule anti-virus scanning tasks to be run at any time, but cannot itself guarantee that the tasks actually will be run.
The main responsibility of F-Agent is to run these scheduled tasks on time. Not all tasks need to be scheduled, but those that do, have two scheduling properties:

- Frequency – daily, weekly on a specified set of weekdays or monthly.
- Time – a fixed time of day or a certain user idle period. Idle tasks are run only once per day, and if the system happens not to be idle for an entire day, the task is re-scheduled for the next day.

As noted earlier, F-Agent typically cannot track user idle time when running as a service. This drawback is effectively eliminated by starting the application-mode agent when a user logs in.

### 2.3 Task distribution

The F-Secure Anti-Virus administrator has the option to automatically distribute scanning tasks. Distributed tasks are copied to the TASKS subdirectory of the shared communications directory and the tasks index is updated. The next time F-Agent on a workstation consults the subdirectory, it will notice that there is a new task available and make a local copy of it.

F-Agent regularly polls the communications directory for changes to make sure that new tasks are distributed quickly. The polling frequency can be set by the administrator and defaults to six minute intervals. To avoid race conditions, a file-based locking mechanism is enforced.

A distributed task can later be “undistributed” by the administrator in a similar manner. The file extension of the shared task file is changed and the task counter is updated to signal the change to F-Agent. The file extensions determine what action F-Agent will take.

<table>
<thead>
<tr>
<th>Extension</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>.FPA</td>
<td>Add a new F-Secure Anti-Virus task to the local tasks</td>
</tr>
<tr>
<td>.DEL</td>
<td>Delete an existing F-Secure Anti-Virus task</td>
</tr>
</tbody>
</table>

*Table 1 F-Secure Anti-Virus task file extensions*
2.4 Software updates

F-Agent participates only indirectly in the software update process. During the poll for changes in the communications directory, F-Agent also notices if an update is available. As it does not contain the logic to perform the update, it launches F-Secure Anti-Virus to do the work.

F-Secure Anti-Virus checks for updates when it is started and takes the appropriate actions to install the updated version. Automatic updating is an essential part of anti-virus software, as new virus descriptions constantly are added to the database.

The model where the application knows how to update itself works when there is an existing installation on the system, but cannot perform the initial installation. F-Secure Anti-Virus solves this problem using a separate utility named AUTOINST that is responsible for first-time installations and also can perform limited updates. AUTOINST is designed to be run in the user's login script. It copies the necessary files to the user's workstation and performs the necessary system configurations. This approach has limitations, as it cannot install files that require higher privileges than those of the user. [AUTOINST]

2.5 Alerting and bulletins

The real-time scanning component of F-Secure Anti-Virus, "Gatekeeper", does not have direct access to the communications directory. If a virus is found and F-Secure Anti-Virus is configured to forward the report to the administrator, Gatekeeper stores the report and the suspect file locally. F-Agent notices that a new report has arrived and copies it together with the suspected file to the network communications directory. As before, access to the network is protected by a file-based locking mechanism, and the appropriate indexes are updated. Virus reports by the on-demand scanner are sent to the communications directory without F-Agent intervening.

F-Secure Anti-Virus provides the administrator with the possibility to distribute bulletins to all F-Secure Anti-Virus users. Typically these bulletins would relate to updates and new tasks being distributed, but there really is no limit to their content. Bulletin files can be of any registered file type and are eventually launched in the appropriate application. The distribution of bulletins uses the communications directory
and a matching index file. When a new bulletin is made available, the index is updated and F-Agent notices the change on the next poll.

2.6 Limitations

F-Agent was originally designed for operating systems with very little local access control. When necessary, virtually any files could be updated without requiring special privileges or costly workarounds. Once the appropriate network privileges were in place, F-Agent worked as expected. This design concept is outdated and was changed as operating systems with access control features grew more popular. The result is a large amount of legacy code and unclear or nonexistent module boundaries.

The division into an application-mode part and a service part solved some of the more critical problems F-Agent faced. The service has the necessary privileges on the local host to communicate with the real-time scanner, but at a cost of not having access to a user interface. The application can communicate with the user, but it is restricted by user privileges.

The current design of F-Agent, combined with the updating properties of F-Secure Anti-Virus, make two a very powerful tool. Software and database updates are automatically distributed to the end-user workstations with a flick of the administrator’s finger. This power also carries the responsibility of preventing malicious users from doing the same. The security level of the current implementation is adequate, but raising it further will require a thorough redesign.
3. Base requirements

3.1 Perspective
The current challenges to the Internet differ slightly from past challenges it has faced. As large corporations have considered connecting to the Internet, they have faced two major obstacles. Administering an internetworked heterogeneous corporate network is far more difficult than administrating a LAN, which often is based on a single proprietary software. Keeping this network secure from intruders, curious employees and malicious ex-workers can be an administrator's nightmare. Adding these two together can mean that the costs of administration and security become prohibitively high. More often than not, however, these problems are quietly ignored, leading to even greater indirect costs. [CSI98] From our perspective, the differences between LANs, intranets, and the Internet are disappearing.

We see a need for automated administrative procedures to lower the cost of administration, and for comprehensive security solutions that provide security without interfering with everyday network tasks. Combining the two should yield a security model that not only provides robust security but also decreases administrative work required. In our experience, this means working at a higher level of abstraction in order to hide the details and isolate the critical components.
3.1.1 Independent operation

A basic requirement for the security model is that it should allow the different functions to operate independently, but cooperate when possible. In heterogeneous systems, only a subset of these functions may be available at a given moment. Relying on some of them to be continuously present could have severe repercussions. With this in mind, a capability-based security model seems to be most appropriate, because it can be designed to be self-contained.

A capability can be viewed as a ticket granting a subject the right to perform some action on an object. Later we will present the concept of an authorization certificate as described in [SPKI97a], [LN97] et. al., and explain why it provides a good base for independent operation. The most important difference between a capability and an authorization certificate is that the former must be kept secret while the latter may be public.
3.1.2 Well-defined interfaces

We aim to incorporate a number of open interfaces into the design. This highlights the requirement for a modular design, as arbitrarily changing the interfaces later is not an option. The same requirement is also true for internal inter-module interfaces, albeit more from a software design point of view. This is often overlooked, even though it is considered self-evident.

The following open interfaces will be defined. The interfaces are described in more detail in Appendix A, and are not defined solely by us. We expect many of the interfaces to evolve when new applications are found. All interfaces can be considered as consisting of two separate parts, an API and an SPI.

![Diagram of Agent interface structure](image)

**Figure 2 Agent interface structure (picture courtesy of Ari Hyppönen)**

- Kernel-mode driver interface. This will provide software vendors with easy low-level access to file functions. Examples are anti-virus scanners and file-archive libraries.

- User-mode application interface. Applications, their user interfaces, databases and application-layer protocols are provided with a user-mode interface for communication with drivers and management functions.

- Key and authorization interface. User authorization and key management is done through a single interface, hiding the underlying implementations.
• Hardware interface. Smart-card readers and biometric identification devices are just two examples of relevant security-related hardware. Due to the device-dependent nature of the provided functions, we may need to supply several hardware interfaces.

• Management interface. We will support a range of management APIs that are provided through a uniform interface. The services provided are intended for network management and are typically accessed from a management console.

3.2 Functions

Later we will describe the different modules of the agent. Together, these modules will perform the following functions, along with potentially numerous other functions that have not yet been conceived.

• Software management. This is a very large area which includes both software updates and managing software configurations. In large networks, this is a critical function and the security requirements are very strict. Only authorized personnel should be able to perform operations related to software management – the implications of unauthorized access to these functions are devastating.

• Reliable authorization and authentication. Traditional systems for identification and authentication have often proven to be the proverbial weakest link in otherwise fairly secure systems. The requirement for authentication has also made many systems unnecessarily cumbersome when the system would actually be better served by reliable authorization. When the identity of the actor is irrelevant or unimportant, we may choose to ignore it in order to simplify the system.

• Identity management. This function may be considered controversial, and not everyone agrees on what it really means. There is an apparent need to separate identities from names, but equally apparent is the need to bind each identity to some kind of name. The issue often boils down to what you consider a "name". We prefer to consider names local in scope, but there is some support for a concept of globally unique names. [SDSI96] describes an infrastructure based on local names, and was later joined with the SPKI effort [SPKI97a], [SPKI97b] and [SPKI97c].

• Policy-based management. Earlier attempts at enforcing system policies have often resulted in complex systems with numerous inconspicuous flaws. We choose to
integrate policy management into the core of the security system, hoping to simplify it enough to make it robust.

- Auditing. A critical part of any security system, auditing provides the necessary fallback when the security measures fail. Proactive monitoring can also raise alerts when an anomaly is detected, be it security-related or a systems failure. At best, this can be used to prevent further damage or to eliminate the cause entirely.

- Cryptography. Even though this function is less apparent to the end-user, it will be crucial for the entire system. Cryptographic certificates are the means with which we will implement policies, authorization and authentication, just to mention a few examples. Several of the other functions also require integrity and privacy that can only be achieved using strong cryptography.

3.3 User roles

The users can basically be split into two groups – those who define policies and those who only follow policies. This division is not clear-cut, however, because policies are layered. A particular user may have the authority to set some system policies, but still must follow upper-level policies set by others. For the sake of simplicity, we shall refer to a user's role at a particular moment, with regard to a certain function, rather than consider all of the user's privileges. In a capability-based system, this is enough, as the authority to perform a function is deduced from the capabilities the user presents.

[Har88]

Administrators have the capability to change system policy and perform administrative functions. This means that they can change the way the agent works and what it does, and thereby influence other users' environments.

Users may have the capability to change some settings that control their own environment, but cannot typically influence the environment of others.

The level of skill we require of users is fairly low. Once a user has mastered the basic concepts, the agent will do its work silently with little user-intervention. This results in a requirement for an intuitive user-interface with a gentle learning curve.

The expertise required from administrators varies according to the task, but can be considerable in some respects. The agent and its management interfaces provide top-
level administrators with almost total control of the system. An error made by an administrator may have instant and far-reaching effects, and the number of sanity checks we can enforce is small. After all, the agent is supposed to be able to update and configure virtually any part of the system.

3.4 General constraints

The initial agent will run on Windows NT 4.0, but care has been taken to make it easy to port to other operating systems as well. The agent does not rely on any unique operating system features, even though some of the code is very specific to the particular system.

Some of the security-critical parts may be implemented in Java to provide a sandbox security model on any platform. This model of security is well suited for an application that has open interfaces, because it reduces the potential damage caused by malfunctioning or malicious applications. As the agent serves a central security role, attacks are both conceivable and probable. The only reasonable approach to reduce this threat is preemption, because the agent handles vital tokens and information. [Amo94]

The agent is a critical component, and it could become a performance bottleneck. To avoid possible future problems, we plan to use concurrent programming and asynchronous techniques whenever applicable. Optimizing performance is also a way to reduce the risk of denial-of-service attacks.

3.5 Assumptions and dependencies

We make a few assumptions that seem reasonable at the moment. Some of these assumptions are critical; if they are incorrect, the security of the whole system is undermined.

Cryptographic certificates require public key cryptography and a corresponding public key infrastructure (PKI). We assume that one or several suitable infrastructures will be available soon. Without a working PKI, the whole system is severely crippled.

Widespread use of public-key cryptography creates a need for secure private key storage. If private keys are compromised too often, confidence in the system will be
lost. We assume that methods to minimize the number of private key compromises are developed and widely deployed.

We are forced to trust the entire operating system. From a security viewpoint this is bad, because it creates a very large trusted computing base (TCB); but today it is reality. We can make this assumption if we claim that it is not a critical one. We are not comfortable with the assumption, but we will have to live with it. Hopefully, future work will reduce the TCB further.
4. Main Concepts

4.1 Identity and names

What is "identity"? Merriam-Webster's Collegiate Dictionary defines identity as follows. [MW94]

identity n, pl. -ties [MF identite, fr. LL identitat-, identitas, prob. fr. L identidem repeatedly, contr. of idem et idem, lit., same and same] (1570) 1 a: sameness of essential or generic character in different instances b: sameness in all that constitutes the objective reality of a thing: oneness 2 a: the distinguishing character or personality of an individual: individuality b: the relation established by psychological identification 3: the condition of being the same with something described or asserted <establish the ~ of stolen goods> 4: an equation that is satisfied for all values of the symbols 5: identity element

The reference to “individuality” is relevant.

individuality n, pl. -ties (1614) 1 a: total character peculiar to and distinguishing an individual from others b: personality 2 archaic: the quality or state of being indivisible 3: separate or distinct existence 4: individual, person

In the traditional sense, it seems that identity is the sum of all the properties that distinguish an individual from others. It is evident from this definition that nobody can know the “true identity” of a person, as this sum consists of an incredible amount of data. Equally evident is that there really is no need to know the “true identity” of a person to be able to distinguish him or her from others – one distinguishing property should be enough. What that property is depends very much on the situation, but in most ordinary situations, it is typically a name.

Everyone is familiar with names. We use them every day to refer to people we work with, live with, or have seen on TV. When we mention a name, our audience associates it with a person and the properties of that person. We have effectively identified the person by mentioning him or her by name. The problem, identified in [SPKI97b], is that this approach does not scale well. When addressing a larger audience, a person in the audience may associate the name with a different identity, without realizing the error.
Now this person will misunderstand everything said and will associate it with the wrong identity (person).

Some digital certificate systems, such as X.509 and PGP, try to address the scalability issue by defining unique global names for every entity [X.509]. The logic behind this is fairly simple; just extend the "ordinary" names with enough information to make them unambiguous to every member of every audience. This way there is no risk of confusion, as the name always identifies a single person. There are two basic ways of making a name unique. Either keep adding information to the name until it is globally unique, or generate a new name using a process that always creates globally unique names.

Long global names must include enough information about a person for everyone to uniquely identify that person. While this approach may initially look appealing, it has its own drawbacks which make it unsuitable for our purposes. If the global name includes a person's common name, people tend to guess at which global name is correct. The name will, in fact, also constitute a dossier about the person, and is thus confidential [Amo94]. Incidentally, this is one of the reasons why the X.500 directory model has never gained widespread acceptance. Corporations who manage their own directories usually consider them company confidential.

This issue becomes even more complicated in a networked society, where relationships may be formed entirely in cyberspace. In that case, there may not be any information that would be meaningful in a long global name.

Short global names must only be unique. This solves the problem of the name being confidential, but it makes it impossible to look up a name based on other information. For the name to have meaning, it has to be bound to some local information. You can compare this to a telephone number scribbled on a piece of paper – without additional information the number is of little value, but with the note, "Laura, the cute girl at the newspaper stand", it is instantly useful. However, at that moment the name is no longer global, it is bound to the namespace of whoever wrote the note. This leads us to believe that short global names are useful only when used in local namespaces.

To be able to communicate a short global name in a meaningful manner, you must find a point of reference that is common to the communicating parties. This essentially reduces to finding a point where the two namespaces can be linked. For most
applications, a public key can serve as a short global name, identifying the person
possessing and using the corresponding private key. Signed certificates about that key
can serve as the point of reference.

4.1.1 Definitions

Until now, we have freely played with the words “identity”, “name”, “global name” and
“person”. To avoid confusion, we define the following terms, freely adapted from
[SPKI97b]. You should note that these terms do not seem to have an established
interpretation. Therefore, these definitions may differ from what you are accustomed to.
In addition, these definitions are technical by nature and are not globally applicable.
Some of the concepts used here are defined later in this chapter.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>principal</td>
<td>A private (signature) key with a corresponding public key. The private key is the only entity that is capable of “speech” and is identified by its public key.</td>
</tr>
<tr>
<td>keyholder</td>
<td>A person or entity in possession of a certain principal. This entity is seldom directly involved. Instead, the principal “speaks for” the entity.</td>
</tr>
<tr>
<td>namespace</td>
<td>A set of names, as defined by a principal.</td>
</tr>
<tr>
<td>name</td>
<td>A token uniquely identifying a principal within a certain namespace, also known as a domain of interpretation. The definition of a name may explicitly refer to names in other namespaces.</td>
</tr>
<tr>
<td>global name</td>
<td>A name in the namespace of the global community. The only global names we recognize are public keys, which identify their principals.</td>
</tr>
<tr>
<td>key</td>
<td>A public key identifying a principal. When we refer to “key”, we typically mean “the principal identified by this public key”. The reason for this indirection is self-evident; the private key is not public knowledge.</td>
</tr>
</tbody>
</table>
certificate A digitally signed record that binds some characteristic to a principal or set of principals.

prover The keyholder that wishes to perform some action or obtain some authorization.

verifier The keyholder that needs to verify the prover’s request for action or authorization.

At this point, we should define “identity”. However, based on the previous definitions, the definition of identity is different from its traditional definition. The difference is explained by the reduced scope. We are not interested in all the aspects of identity. Some interesting side-effects of this change of scope are reviewed in the following chapters.

identity The least amount of information required to unambiguously associate a principal with a given characteristic. Usually, this is simply “key”. It should be noted that this definition is valid in the scope of a single authorization and loses its meaning when applied to a broader context.

4.2 Authentication vs. authorization

Traditional systems employ a two-phase access control strategy. The person requiring access to the system starts by identifying himself. This can be done in a number of ways, but typically it involves feeding the system some kind of user identifier. The system then uses this user identifier to check what kind of authentication procedure is required for access using that particular identifier.

The purpose of the authentication procedure is to verify that the given identity is correct, i.e., that the person presenting the identifier is actually the person it identifies. This is necessary to (temporarily) assign the person the privileges given to the identity in question. Once authenticated, the user can act using those privileges until he relinquishes them.

The different authentication procedures can be divided into three main categories, *something known, something embodied* and *something held* [Amo94]. When stronger
authentication is required, procedures from several of the categories can be combined making the work for a potential attacker considerably more difficult.

![Diagram of Access using identification and authentication]

**Figure 3 Access using identification and authentication**

With the advent of the Internet and a growing networked economy, the traditional approach presents a risk to privacy. As the user is required to give his or her identity to the remote system to use the provided services, the system is able to create a user profile based on what the user accesses. When several such profiles are combined, a detailed dossier can be built, which tells the reader more about the user than the individual parts. This risk should not be trivialized, as similar practices are in wide use in other fields today.

The diverse nature of the Internet also means that users need to have access to a large number of completely separate services. Each service requires its own (identity, token) pair and using the same identity or token in separate services is a clear security threat. Nevertheless, users tend to use a small set of identities and tokens, which means that if one service is compromised, others are compromised as well. Identifying the other services that may have been compromised is difficult if not impossible.

Fortunately, there is an alternative to identity-based authentication. Many services require authentication to prevent unauthorized use, often to ensure that only paying customers can benefit from the services provided. In view of this, it would be conceivable that most of the services could operate equally well, if the clients simply could prove that they are authorized to use the service. This proof can be presented in the form of an authorization certificate chain.
In its simplest form, an authorization certificate is a signed record delegating some authorization to a principal [SPKI97b]. When the principal wishes to access a service, he presents the appropriate certificate to the service. To establish that the request is issued by the correct principal, the service requires that the principal signs a one-time challenge before allowing the principal to proceed. This differs from authentication in that the service need not know anything about the principal except its public key. If this public key was created for the sole purpose of using this service, the transaction is, in essence, anonymous and cannot be bound to anything else.

![Diagram of Access using an authorization certificate](image)

**Figure 4 Access using an authorization certificate**

Authorization certificates are an extension of capabilities. Capabilities have many of the same properties, but must be kept secret at all times, as possession of a capability is enough to gain access to a service. A capability-based system has many advantages compared to a traditional system and solves some tricky access-control problems [Har88] [Lan89].

### 4.3 Trust

Trust can be seen as something you know. This is opposed to information, which is something you receive and did not know beforehand. The only ultimately trusted source for you is yourself. Every other trust relationship contains the chance of a surprise, a certain risk. You judge information you receive based on prior knowledge and decide if you can trust the information or not. From this follows that there is no universal foundation of trust that everyone could build on; trust is purely subjective. And, as you generally trust yourself, trust is reflexive.
Trust has some interesting properties that are relevant to our work. Unfortunately, these properties are counter-productive to what we want to achieve, which means that we have to make do with a best-effort solution.

Trust is not transitive. If you trust Alice, and Alice trusts Bob, this does not mean that you automatically trust Bob. As a matter of fact, you may actually be of the opinion that Alice generally has extremely untrustworthy friends. You might be able to trust Bob to a certain extent, perhaps to take out the garbage, but would not let him water your plants while you are on vacation. We need a mechanism that allows you to express different levels of trust.

Trust is not symmetric. You trust Cecile to take good care of your plants. As a matter of fact, they always seem to blossom while you are on vacation and fade away quickly once you return. If you ask Cecile, she thinks you don't have a clue about taking care of plants and would never let you water her plants while she’s away on vacation. This really is no problem, as long as we remember to account for it.

Trust is not distributive [Ger97]. You may trust Cecile to water your plants and give her your key, but had you known that she was seeing Bob at the time, you would have chosen somebody else. Solving this problem is hard, as you may not foresee all the possible problems that can arise. As a matter of fact, Cecile may start dating Bob only once your vacation has started, which means that at the time, your trust was not misplaced. We intend to address this by using the principle of least privileges. We hope to reduce the negative side effects by minimizing the set of unknown preconditions.

The combined effect of these properties is that trusting someone is always a risk. The risk grows significantly if the trust relationships are chained one after each other, according to the rule of combined probabilities. Such chains of trust are very usual in all environments and can often lead to surprising outcomes. [Tho84] From a technical point of view, a certificate chain is a number of delegations of authority from one principal to another. It cannot automatically be assumed that such a chain reflects trust in the subject.

4.4 Delegating authority

Before going on vacation, you gave Alice the key to your apartment to allow her to water your plants. Your intention was that she was to water the plants herself, but you
have no way of controlling that. If Alice gives your key to Bob, she abuses the implicit trust you placed on her not to delegate the task of watering plants. To make matters worse, Bob may abuse Alice’s trust by emptying your freezer while watching your favorite videos on your VCR. The former problem is caused by the fact that it was possible for Alice to delegate the authority placed on her, the latter by the fact that you had no way of delegating the least privilege necessary to Alice.

The use of authorization certificates allows a virtually unlimited precision when delegating authorization. The certificate expresses the authorization delegated to the subject with the required precision. In this, authorization certificates differ considerably from the above example. The authorization is not implicit in the key; it is expressed explicitly separately from the key and bound to the key of the subject.

When you hand over a physical key, you delegate access to the entire space controlled by that key without discrimination. The same happens if you hand over control of a subject key, a principal – you effectively delegate all the authority that the key has. To avoid situations where principals are passed between keyholders, we need to allow delegation of authority where it is appropriate. When you go off on your vacation, you may choose to delegate the authority to sign your bills to your secretary, for the duration of the vacation. However, you would probably not allow your secretary to further delegate that authority. Therefore, allowing delegation also calls for a method to restrict delegation. The compromise between delegation and lending of principals is one of the challenges of this model. The chance of someone lending a principal instead of using delegation always remains.

The example of delegating the authority to sign bills demonstrates an interesting property of authorization certificates. The certificate is re-used whenever needed, therefore your secretary could use it repeatedly to empty your account. Restricting the total amount to be spent requires more advanced functionality, where some kind of memory is needed to store the amount spent so far. This memory has to be independent of the certificate, as you can make any number of copies of a certificate. The way such memory should be accessed by the verifier is an example of policy. If the total amount has to be checked with every transaction, off-line transactions are impossible. If the amount has to be regularly re-verified, fraud is possible by exceeding the total amount through repeatedly using the certificate before the next re-verification is scheduled. The concept in itself is not new, but when using authorization certificates, this policy can be
recorded in the certificate, or a reference to it can be included in the certificate. This makes the verification process much easier, as the verifier does not need to locally store all possible policies.

Authority, like trust, always originates from the issuer. When using authority certificates with delegation, the certificate verifier needs to form a certificate loop to accept the authorization, that starts and ends in itself [LN97]. It is important to understand that the certificate loop is not a trust loop. The first delegation step is an expression of trust by the issuer on the subject at the time, but the remaining steps are not previously known to the issuer. The non-distributive property of trust implies that these steps do not express trust by the issuer on the subjects of the certificates. When the issuer thus takes on the role of the verifier it has to decide to trust the certificate loop or reject the authorization request. This trust decision is based on the authorizations or policies expressed in the certificates, but may also involve local policies external to the certificate loop. As trust is also a function of time, even the original statement of trust may be revoked. Thus, presenting a valid authorization certificate loop to a verifier does not automatically imply that the authorization is granted.

![Diagram of authorization loop](image)

**Figure 5 Authorization loop, not trust loop**

### 4.5 Policy

[Bla96] describes a language that can be used to express policies in a system and a framework where applications can use PolicyMaker to evaluate trust queries. The policies are assumed to be stored locally, either as inherently trusted statements or as policy statements by principals. An interpreted language is used to evaluate the policy statements, something which may require a confined environment imposed by the
language or its execution environment [Lam73]. This approach requires lookups in a policy database, which could potentially become a performance bottleneck in complex systems with a large number of policy statements.

[SPKI97a] takes a slightly different approach, where policy statements are included in authorization certificates. The current definition is not as versatile as PolicyMaker, but [SPKI97b] notes that it is possible to combine authorization certificates and PolicyMaker when reducing authorization loops. This approach would reduce the size of the required policy database, as part of the policy statements can be carried within the authorization certificates.

Policy can be seen as a means to control the choices available. Whenever a subject wants to perform an action on an object, policy states whether this is allowed or forbidden. Policy is exact – it can be seen as a function \( f \) from the cross-product of subjects, actions, and objects to a boolean value. This is the operation of a reference monitor [Amo94].

\[
f: \text{subjects} \times \text{actions} \times \text{objects} \rightarrow \text{boolean}
\]

In traditional systems, the policies have often been expressed informally. Often the actual function of the system has reflected only some of the policy. We intend to make policy-based management a core component in the agent. All actions have to pass through the policy manager, which approves or disapproves the actions according to the installed policy. Here the concept of "action" is broad, including not only actions by the user, but also actions by the user-interface and agent modules. Thus, the policy may prevent the user-interface from showing certain user-interface components altogether or prevent a loaded module from performing some undesired actions. This will allow us to use a single application for all types of users, an application that will dynamically alter its appearance and functionality according to policy. Currently, this customization is done during installation, hard-coding the policy in the workstations.

4.6 Cryptography

Cryptography is the art and science of encoding messages to ensure that a desired property is retained. Together with cryptoanalysis, the art and science of decoding enciphered messages, they form the science commonly known as cryptology. We will base our work on functionality that is best achieved through the use of cryptography,
but neither go into the actual details of the ciphers used nor the cryptoanalytic methods that could be used to attack them. We are interested in the cryptographic services that can be utilized in a secure system and assume that you are familiar with the basics of cryptography. Details about the algorithms referred to can be found in [Sch94].

Cryptographic services are often arbitrarily divided into basic and advanced services. The advanced cryptographic services are built using basic services, mostly in the form of cryptographic protocols designed to perform a particular task. The basic services are **integrity, confidentiality** and **authentication**. Advanced services are **authorization**, **anonymity** and **non-repudiation**, among others. The design and implementation of cryptographic protocols is an art that only recently has been evolving into science [Nik98].

### 4.6.1 Basic services

Integrity, or rather the ability to detect breaches in integrity, can be provided using cryptographic hash functions. A hash of the original data is stored separately from the data itself. Later, when the integrity needs to be verified, the hash is re-calculated and the two hash values are compared. If they do not match, the original data has been modified. Good cryptographic hash functions have the property that given h(x), it is very hard to find an y such that h(x) = h(y). This, of course, includes the fact that given h(x) it is very hard to find x itself.

Confidentiality can be provided using a cryptographic algorithm known as a cipher. In modern cryptography, the algorithm itself is considered public. The entire secret rests in the key used in the encryption process. Symmetric ciphers use the same key for encryption and decryption, whereas asymmetric ciphers have two different keys for the operations. The asymmetric ciphers have interesting properties, which are at the core of modern cryptosystems. The encryption key is typically referred to as the public key, and the decryption key is known as the private key. When used in the opposite direction, the private key serves as a signature key and the public key as a signature verification key. This is the main property that makes asymmetric ciphers so important.

Authentication is the ability to verify that messages originate from a given source. This should not be confused with nonrepudiation, which in addition provides proof of the authenticity to a third party. Authentication can be achieved using keyed cryptographic hash functions. Only the party or parties in possession of the correct key can generate
and verify the keyed hash values. An attacker who modifies the data cannot do it without invalidating the keyed hash value. Authentication thus also implies integrity, but should not be confused with checksums frequently used to detect random errors in data, as the checksums can be generated and verified without a key.

### 4.6.2 Advanced services

The definition of advanced cryptographic services often depends on the context in which they are used. We are interested in the following definitions.

A digital signature is a hash of an object encrypted with a private signature key. The signature can be checked by decrypting the digital signature using the public signature verification key, yielding the original hash value, and then calculating the hash value of the signed object to compare the two hash values. If they match, the original signature was made using the named private key and is therefore valid.

Authorization is the ability to prove that one is authorized to perform a given action. It is possible to do this without disclosing a meaningful identity, which makes authorization suitable for scenarios where identity-based authentication is used today. An authorization must unambiguously identify both the object of the authorization and the action the principal is authorized to perform on the object. The principal can prove its identity using a digital signature, but does not need to disclose anything more about itself.

Nonrepudiation, in the cryptographic sense, means the ability to prove to a third party that a message originated from a given source without requiring cooperation from the source itself. In its simplest form, it is a digital signature, as a signature uniquely identifies the principal used for the signature. Authentication is not enough, because the key used for authentication may be shared between the communicating parties. Thus, any one of the keyholders may have been the source of the message. Cryptographic nonrepudiation is quite different from the legal concept of nonrepudiation and should not be confused with it. It may be cryptographically possible to prove that a certain principal was used, but legally impossible to bind any keyholder to the use of that key.

Anonymity can be considered an advanced service, although a controversial one. For anonymity to be useful, it is necessary to use cryptographic authentication to link separate messages from the same anonymous source without revealing anything except
the public key of the source. It must be nearly impossible to trace the messages to their source, but still possible to communicate with the source itself. With our narrow definition of “identity”, anonymity means the ability to act in the role of one identity without disclosing information about other identities one may have. This is a two-edged sword because it can serve to hide good as well as evil.

4.6.3 Cryptographic certificates

A certificate was previously defined as a digitally signed record that binds some characteristic to a principal or set of principals. The interpretation of this characteristic depends entirely on the verifier; thus the semantics of a certificate need to be defined before it can be used.

An identity certificate maps a name to a principal. It answers the question, “What principal is associated with this name?” In itself, this is of little use, which is why a further step of binding authorization to the name or the principal is necessary. This approach has several weaknesses, as explained in [SPKI97b] and [Leh97]. It is important to notice that the mapping is one-way, thus assuming that the principal automatically maps to the name is not correct. If such a mapping is desired, it has to be made explicit.

An authorization certificate binds an authorization to a principal. It answers the question, “Is this principal authorized to perform this action?” This is the functionality provided by a reference monitor, and what a security policy expresses. Thus, authorization certificates seem to be the ideal tool for a policy-based management agent because they express policy, and their evaluation performs the function of a reference monitor. We will concentrate our effort on SPKI certificates, as defined in [SPKI97a], [SPKI97b] and [SPKI97c]. The most important difference between policy and authorization is that authorization always contains the concept of validity. An authorization certificate used outside its validity period grants no authorization.

\[
f: \text{subjects} \times \text{actions} \times \text{objects} \times \text{validity} \rightarrow \text{boolean}\]

According to [SPKI97a] an SPKI certificate can be expressed as a 5-tuple \((I,S,D,A,V)\) where
I = ISSUER a principal or a single, top-level name in a principal's namespace. The principal is identified as a public key or the hash of that key, and the corresponding private key signs the certificate.

S = SUBJECT a principal, an object or a SDSI name reducible to either of them. The subject is that which receives authority from the issuer by way of the certificate.

D = DELEGATION the optional modifier, "(propagate)" , giving the subject permission to delegate the authority presented in the certificate (or part of it) to some other Subject. This is represented as the delegation boolean D. The two boolean states are (F: delegate only through name declarations – also known as "stop at key") and (T: delegate to the subject).

A = AUTHORITY the specific authorization(s) being delegated in this certificate. These fields, in the form "(tag ...)", are to be defined by those who have resources to control and describe resource allocations.

V = VALIDITY date ranges and/or online validity tests for determining certificate validity.

When presented with an authorization certificate or a chain of certificates, the verifier has to make sure that I is "Self", i.e., the verifier's own key. For a single certificate, this is trivial, but for a chain of certificates the verifier has to perform a 5-tuple reduction.

The reduction of (I1,S1,D1,A1,V1) + (I2,S2,D2,A2,V2) into (I1,S2,D2,A,V) [SPKI97b] is simple:

if S1=I2 (meaning that they are the same public key)
and (D1 = TRUE) or (S1 is a SDSI name)
and A = intersection(A1,A2)
and V = intersection(V1,V2).

If I2 is a SDSI name or S1 is a threshold subject, the process is more complicated, but eventually yields the expected result or fails if the chain is not complete.
The authority field has a fixed syntax for specifying authorizations and several examples are listed in [SPKI97c]. The exact contents of the authority field are application-specific, and their meaning is defined by the verifier.

The validity field supports both off-line and on-line tests for validity. The off-line test is simply a date range expressing the validity of the certificate. The on-line test allows the use of a certificate revocation list or the re-validation of the certificate for a specified period of time. The re-validation can also be one-time, where the verifier sends a nonce and receives a signed nonce in return if the certificate is to be considered valid for the duration of this computation.

The actual syntax of an SPKI certificate is a restricted form of S-expressions. The format is binary, but a presentation format is defined for the contents of a certificate to be presentable in human-readable form. Only the binary format is used in implementations.

Names in SPKI certificates are always local to a specific principal. The 5-tuple reduction allows authority to be arbitrarily delegated through names, as they only serve to connect principals to each other in a late-binding fashion. Delegation from one principal to another is only allowed if the D field is present. There are several advantages in using names, due to the late binding they introduce. A good example is the replacement of a lost principal, which could potentially be a very difficult task if the principal had a large number of authorizations delegated to it. If the authorizations are delegated to a name instead, all the authorizations could be restored by issuing a new certificate binding the name to a new principal created to replace the principal that was lost.

The use of local SPKI names does not introduce the same lookup risks as the use of global distinguished names in X.509, as the names are local and reduced automatically. However, if necessary, the X.509 namespace can be considered to be a local SPKI namespace to allow delegation to use distinguished names. Other similar namespaces, such as PGP or DNS, can also be expressed as SPKI local namespaces.
5. Structure

5.1 Framework & technology

The architecture of the agent is influenced to some degree by the current F-Agent structure. Some of the problems the agent is intended to solve are inherited from F-Agent, which is why we reuse the features that still work on current problems. However, where F-Agent falls short of its goal or simply does not have similar functionality, we introduce new solutions. Even when reusing a feature, we intend to completely rewrite it from the bottom up to dispose of legacy code and create a more modular design.

5.1.1 Kernel-mode drivers and open APIs

At the core of the agent is a group of kernel-mode device drivers. These drivers allow us to intercept file system calls to perform anti-virus scanning and encryption in real-time.
as part of the operating system. Open APIs are provided to third-party vendors who can use the APIs to extend the functionality of our drivers with modules we call "plug-ins". The APIs are considerably simpler than the general device driver interface. The definition of the APIs are beyond the scope of this work. Their current state is described in Appendix A.

Defining an open API gives us the possibility to enforce a policy on the calls to the API. Part of the policy-based management resides in the kernel-mode drivers, but performance issues restrict the functionality that can be provided in the kernel. Whenever the kernel driver faces a more complicated policy evaluation, it must call the user-mode components to evaluate the policy. The monolithic structure of most current kernels would otherwise block all other system activity until the policy evaluation was completed. Bringing the evaluation to user-mode allows the operating system scheduler to preempt the policy evaluation when necessary. Of course, the kernel-mode driver cannot complete outstanding requests until the policy evaluation completes. Therefore, we expect to run the policy evaluations at a high priority level.

Implementing kernel-mode device drivers requires a high level of skill and exceptional code quality. To keep the complexity of the drivers at a minimum, all tasks that do not have to be done in the kernel for performance reasons will be moved to the user level. We also expect third-party plug-ins to conform to this programming practice. To make this possible, we define a similar API at the user level and also provide an abstract communications channel between kernel and user modes. This approach frees independent programmers from the need to implement communication over the boundary using more complicated device I/O calls.

5.1.2 User-mode process

In contrast to F-Agent, we always need a user-mode process running to provide the device driver with necessary user-mode services. The user-mode process runs as a service even when no user is logged in, but does not contain user interface features. The user interface is provided by a separate process that is started when a user logs in. The advantage of this approach is that the two processes can run with different privileges and in separate memory spaces. This provides the necessary insulation to prevent malicious user-mode applications run by the user from bypassing the security measures and policy management of the agent.
5.1.3 Policy manager

Applications that wish to use the policy-based management module of the agent must delegate part of their functionality to the agent. All managed objects are given unique object identifiers. Whenever an action is to be performed on a managed object, the agent is queried for the policy. The action on the object is authorized only if the user has a matching authorization certificate for it. Examples of this are given in section 5.2. It should be noted that the choice of managed objects depends on the author of the application, but once an object is subject to policy-based management, the behavior of the application depends on the authorization certificates of the user. If a certificate expires, the behavior may change dynamically without user intervention.

5.1.4 Management interface

The same object identifiers used for policy-based management may also be used to interface with third-party management systems. The agent’s management module and applications using the agent can collect and report statistics and report relevant settings to a management system through standard protocols such as SNMP. The selection of statistics and settings available for third-party management systems is also a matter of policy, as some of them may contain potentially very sensitive information. Sensitive information will be accessible only through secure channels that are properly authorized and will never be transmitted in plain-text over a network. The API to the management module is open for third-party implementations to provide support for installed management systems. However, only systems capable of producing valid authorization certificates can change the installed policy; others are limited to read-only access.

5.1.5 Integrated auditing

The agent and its client applications also use the management module for auditing and alerting. All audit events are reported using the same API and are thus subject to the policy enforced by the agent. The agent may decide only to log an event locally, but system policy may dictate that one or several alerts need to be issued. As the agent also does audit reporting internally, alerts may be triggered whenever the agent performs either an action or a policy evaluation on behalf of an application.
5.1.6 Authorization agent

Internally, the agent uses a cache of authorizations derived from reduced authorization certificate chains to efficiently make policy decisions. When a user logs on, the user-specific parts are initialized to provide a single sign-on facility. System policy may restrict the type and validity of entries in the single sign-on cache to avoid potential security risks caused by an unguarded workstation. The client applications interface with the authorization module to answer authorization challenges from third parties. This makes it possible for the user to use authorizations without having to actively participate in the challenge-response sequence. Some of the authorization certificates in the cache may be derived from other certificate systems, such as X.509, provided that the user has indicated what kinds of challenges are to be answered with those certificates. Without user-supplied information, external certificates are of limited value.

5.1.7 Task scheduler

Applications can schedule tasks with the agent for later execution. Initially, this will be used mostly for anti-virus scanning; but reliable authorized scheduling is a service that we expect several third-party applications to benefit from. In large networks, scheduling can also be used to reduce network load peaks often caused by software updates. Authorization for automatic updates is given for different validity times for different systems, thus balancing the load over a period of time. The agent has sufficient privileges to update system components and to run installations with system privileges. These privileges, combined with strong authorization, makes it easier for administrators to automatically update software on workstations without having to fear that attackers could do the same to install malicious code.

5.1.8 Hardware abstraction

Finally, the agent will abstract several complicated interfaces. The authorization module contains a hardware abstraction layer (HAL) for different authentication and identification devices. The cryptographic module provides a uniform interface to both software and hardware implementations of cryptographic algorithms. The agent itself abstracts the complicated device I/O interface for its client applications. All of these interfaces are open to third-party implementations to ensure that once a compliant plug-in is installed, all client applications can immediately utilize it through the same API.
5.2 Policy-based management

The concept of policy-based management depends on who is defining it. We have designed the agent with the reference monitor model in mind, keeping the policy manager at the absolute core of the agent. Thus, we ensure that all actions that are performed through the agent are subject to policy. Only actions allowed by the system policy are possible; all other actions will fail unconditionally. The beauty in this approach is the fact that the system policy is enforced dynamically in real-time. In theory, the behavior of an application may change in front of the user’s eyes when new authorizations are issued or old authorizations expire. In practice, applications will contain critical check-points at which policy changes take effect. This will require some new skills on the behalf of the software designers.

5.2.1 Candidates for policy-based management

The most important class of objects that needs to be managed through policy is the settings of installed applications. For each setting, a number of attributes can be defined.

- Type, including all relevant information about the setting so that it can be manipulated and altered.

- Default value, i.e., what value is assumed if none is defined.

- Allowed values, where the type does not specify the values. This can be, for example, an allowed range of integers for an setting of type Integer.

- Access privileges, i.e., read and write access to the setting.

The type of a setting is defined by the application itself and is not subject to policy management. However, the default value, range of allowed values, and access privileges are clearly something that system policy will define, within the bounds of the application. The application essentially delegates the right to set the value within the reasonable application-dependent limits to the administrator of the policy domain. The administrator may then choose to further delegate this right to users within the domain, as authorization certificates.

The concept of “delegation”, when applied to an application, is interesting. In theory, the designer of the application could issue a certificate which explicitly describes the legal bounds for a setting. The software license could then be a collection of these
certificates, and the application would not function without them. In practice, most of the bounds checking will still be performed by the application, not by the policy manager. We do not believe that requiring explicit certificates for settings is a reasonable approach at this time.

Calling a function of the agent to perform an action requires an appropriate authorization certificate. In general, the agent itself will have a copy of the required certificate in its cache, thus removing the overhead of continuously passing fresh authorization certificates with the function calls. The certificates for accessing the agent are installed by the same routine that installs the application requiring the certificate. Conceptually, they are no different from other authorization certificates, but in practice, they simply serve to verify that the application being installed is approved for installation and use. Nevertheless, a reference to the correct authorization certificate must always be included in the function calls, typically a hash of a certificate result certificate. In practice, authorization certificates are internally reduced to capabilities to decrease the overhead of internal function calls.

Even the right to read the value of a setting requires authorization. The policy manager will refuse to provide the application with a value it has no right to access, but for most settings, the application will need to have at least read access to function properly. However, the user may not have authorization to view a setting that the application knows. In those cases, the application may either simply hide the value from view, or more advanced applications may choose to completely eliminate the corresponding user interface elements.

At this point, it is relevant to note that the application has delegated the access control decisions to the agent. Therefore we do not need to guard against applications that intentionally “cheat” by leaking information to unauthorized users. More importantly, however, applications are unable to change the settings in violation of policy, short of creating local copies of them.

5.2.2 General policy types

An example will clarify how different policies affect the applications. Let us assume that there is a setting, “SecretDirectory”, which is of the type “AbsolutePath”. The administrator may define a global default policy, stating that the default value of
SecretDirectory is \texttt{C: \%username\%\Secret} and that normal users do not have any access to this variable.

When an application tries to access SecretDirectory, the agent verifies that the application has authorization to do so. This authorization certificate would typically have been installed with the application itself. An authorized application can read the value and use it for whatever purpose it was intended. The application will probably not have permission to alter the value by default, so any attempts at changing the setting will cause an error and possibly trigger alerts. The application’s user interface will ask for permission to show the setting to the user, but will be denied. If properly designed, the application will not even show that the setting exists.

If the administrator later decides that security managers should have the right to read and change the SecretDirectory setting, the administrator issues the appropriate authorization certificates and distributes them to the managers. When a security manager consequently logs on, the authorization certificate gets fed into the authorization agent. Now the same application, which earlier had not even shown the setting, will note that the agent clears the setting for both read and write access and shows an edit box with the current setting.

For authorization certificates to serve their role in avoiding the Confused Deputy problem, the applications are required to present the appropriate certificates when performing an action requiring authorization [Har88]. Thus when accessing a setting for internal use, applications will present their own certificate. When accessing the setting for user access, they will present the user’s certificate. This leads to the interesting property that a correctly written application may not be able to modify a value by itself, but its user can use the application to change the value.

5.2.3 Managing policy management

One more issue relating to policy-based management deserves attention. Settings can be altered using the correct authorization certificates, but the correct generation of such certificates cannot be managed in quite the same manner.

To be able to generate a valid authorization certificate for an action, one of two things are required: (1) direct control, i.e., ownership, of the object in question; or (2) an authorization certificate with the right to further delegate the authorization. Ownership
means that the issuer is the source of authority for the object and ultimately performs all verification of authority for the object. Anyone can claim to be the owner of an object and issue false certificates, but their value is zero, as the real owner immediately notices that the issuer is incorrect when it attempts to verify the certificates.

From a narrow perspective, the user that owns an object, or has the right to delegate authority over the object, acts in the role of an administrator for that object. We intend to implement a tool that, given an appropriate object identifier or a certificate for that object identifier, generates authorization certificates. In its most basic form, this tool is simply a button in the agent's certificate browser. In its most advanced form, it can be integrated into any application that processes object identifiers, e.g., network management clients.

5.3 Authorization agent

The only way to improve the usability of a complex security system is to provide users with a single sign-on facility. The agent includes a module frequently referred to as an authorization agent. This authorization agent authenticates the user during login and unlocks the user's authorization certificates and authentication tokens. System policy dictates which certificates can be unlocked in this manner and for how long they may be left in an unlocked state.
5.3.1 Agent structure

The agent consists of an authorization certificate cache, an authentication token cache, and an authentication method abstraction layer. Authentication methods and authorization handlers are registered with the authentication agent as separate plug-in modules. The API for the plug-ins is open, so third-party authentication devices and methods can easily be integrated.

The authorization agent itself does not know the particular details of the different authentication dialogs and authorization sequences; it simply serves as a centralized manager of tokens required in the processes. Policy can also dictate which methods are available for different services, which makes it possible to enforce strong authentication when required.

5.3.2 Authentication requests

When a client application receives an authentication request from a service, it calls the authorization agent and identifies the service and request type. The authorization agent consults its policy database to see which authentication method the request should be directed to. It then creates an authentication session and forwards the authentication request to the appropriate method.
The authentication method resumes the authentication dialog with the remote service from where the client application left off. The exact division of work between the client application and the authentication method is method-dependent, but ideally the authentication method can drive the service into an authenticated state without consulting with the client application. Currently, we have not implemented any such methods, but some are described in [SASL97] and [GSSAPI93].

If the authentication method requires an authentication token, such as a password, it needs to ask the user for it. Once obtained, the authentication method may choose to store the token in the authorization agent’s token cache for later use. This way the token can be re-used or even locked away for use in a later user session. Which authentication methods and tokens can be stored this way is subject to policy. Without the correct certificate, the authorization agent will not accept a token for later re-use.

5.3.3 Authorization requests

It is important to separate the different kinds of authorization dialogs that the agent faces. The cache lookups are different and different parts of the agent are involved.

- External authorization challenges are received by some client application in response to an authentication attempt and given to the authorization agent for processing. These require the agent to respond according to a protocol. Figure 8 and Figure 9 describe two such scenarios.

- Internal authorization indications are received by the agent when a client application attempts an action requiring authorization. These require the agent to determine if the application or user is authorized to perform the action, possibly prompting the user.
When a client application receives an authorization request from a service, it calls the authorization agent and identifies the service and request type. The authorization agent reacts in the same way as with authentication requests, locating the appropriate authorization method to serve the client.

The agent also implements a simple authorization response method, where the agent returns an authorization certificate chain to the client to be forwarded to the service.
service will then issue one or several authorization challenges that the authorization agent answers on behalf of the client.

It seems quite likely that some of the authorization methods will again call the agent to perform authentication operations to avoid duplicating their implementation. Digital signatures are authentication methods that can be called from the authorization methods. This way, external plug-ins that provide digital signatures, for example, can be used by all authentication methods.

5.4 Software updates

In a large network, software management quickly becomes a problem. When a new version of an application is released, it needs to be updated to all the computers on the network. If the update cannot be distributed automatically, administrators will have to spend a considerable amount of time installing the new version on workstations or making sure that users do the installation themselves. The same problem, although in a slightly different form, occurs when a new application is to be installed on the network.

In small to medium sized networks, installation can be done when a user logs on to his or her workstation in the morning without performance problems. However, at a certain size, the point is reached where there are enough concurrent installation attempts to either overload the server or saturate the network. At that point, a more advanced distribution strategy has to be set up. The usual approach is to increase the number of servers, which only postpones the problem, offering no scaleable solution.

5.4.1 Large batch updates

The agent scheduler allows an administrator to schedule an update at any time. The agent itself has the necessary functionality to obtain an update package from a network server and to launch it with appropriate system privileges. This means that an application can be installed on a workstation using administrative privileges even when nobody is logged in. From a network administrator’s point of view, this is ideal, because every workstation running the agent can thus be managed remotely.

The scheduling API is part of the open plug-in API. It basically allows an application to specify that a callback should be called at a later point in time, with a given set of parameters. The agent implements a “fetch and run package” callback that is used to
perform software updates. Of course, this callback is only available to users with the appropriate authorization certificates.

The issue of performing a large update operation now becomes a question of how to coordinate the update operations. The administrator can use a network management tool to assign workstations to different groups and schedule the updates separately for the groups. Once the scheduled tasks and necessary authorization certificates have been distributed to the workstations, the updates will be performed automatically by the individual agents on the workstations. The update rate and grouping can be customized to place only a moderate load on the servers and the network, or to run at night when network traffic is slow.

The installation procedure fetches the named binary and verifies its authenticity and required authorization. It then launches the binary and assumes that it performs a reasonable installation. This independence of installation method makes this feature useful for a broad range of applications. As a matter of fact, the fetched application need not even be an installer – it may do something entirely different.

Having an automated software installation and update procedure does not solve all problems related to updates. If the new version of an application is only partially compatible with the old version, the administrator must take this into account when performing the update. If the update does not reflect the workflow in the company, data created by the new version may be sent to users whose workstations have not yet been scheduled for updating. There is little the agent can do to prevent such situations from occurring, which further emphasizes the need for good planning.

5.4.2 Security issues

The introduction of an agent with automatic installation privileges on every network workstation introduces a potential security risk. There are several security threats that have to be addressed.

- Distribution of malicious code. The agent only knows that the installation is authorized. It does not know that the software being installed does not present a security threat. If the installation package contains malicious code, such as a computer virus, the code will be replicated to all workstations in the network. Cleaning up such a major attack is extremely time-consuming and often very
difficult. The most secure way of installing an application requires that the package is signed by its author, ensuring that the package has not been modified since it was shipped.

- Unauthorized distribution of code. If the system policy regulating the update functionality is too lax, it may be possible for users to gain additional privileges using the agent’s higher privileges. Users that do not have the right to install software on their workstations may be able to use the agent to do it. This issue can only be solved by properly educating administrators, to ensure that the update privilege is not delegated too freely.

- Unforeseen side-effects. The software installed may work as expected on 99% of the workstations where it was automatically installed, but cause the remaining 1% to break down. On a 10,000-computer network, that would mean that the administrator suddenly has 100 angry users that need immediate assistance. Had the installation been done manually, the administrator would probably have been able to identify the problem after a few failed installations.

The proper use of authorization certificates in accordance with the principle of least privilege should effectively minimize the risks involved. Any administrator of a large network is prepared to face the slightly increased risk when it is offset by the huge benefits introduced by a comprehensive automated software update system.

### 5.5 Configuration management

Until now, we have simply assumed that there is a way to distribute certificates without describing it in detail. The certificate distribution system is a critical component in the agent, as it provides the agent with the system policies it is supposed to enforce. As the policy is defined in the form of authorization certificates, the agent will deny any operations that are unauthorized. Thus, the lack of certificates will make the agent deny almost all operations, a highly undesirable result.

The certificates authorize actions on object identifiers, which also must be defined for the agent to be able to enforce policy. If the agent is presented with a certificate for an object it does not know about, it is forced to deny the action as it has no knowledge of how to perform it. This is despite the fact that the certificate itself may be valid.
5.5.1 Configuration chaos

There has been numerous attempts at solving the issue of configuration management. The Simple Network Management Protocol (SNMP) is currently in its third phase, often referred to as SNMPv3 [SNMP98]. This protocol defines a standard syntax for configuration data and access control to the data. The object identifier (OID) is a central concept in SNMP, as it refers to every entity it handles by its OID. The information regarding the OIDs of applications is collected into a Management Information Base (MIB).

We have chosen to name the objects we manage in accordance with the SNMP practice and to collect them into a MIB of the managed products. When an application is installed, its MIB information is merged with the information already present on the system. The agent uses the MIB to control the settings of the client applications it manages, but replaces the SNMP access control schema with authorization certificates. To access an OID in any way, a client application must present a valid authorization certificate chain that the agent can reduce to a valid 5-tuple. The result of the access is cached for later access to avoid unnecessary overhead from passing around certificate chains between the agent and its clients.

The MIB format also allows us to describe some other features, such as alerts, which are presented later. The MIB describes the static structure of the different objects, whereas we use authorization certificates to control their dynamic behavior.

5.5.2 Management framework

The agent's management module is an open API to a number of management systems. The agent uses the API for several tasks.

- Information retrieval. The agent can use the management module to consult the management system. This way, missing certificates or MIB definitions can be retrieved and installed locally. In effect, the agent will notice when it has no installed policy and attempt to obtain the policy from a server.

- Reporting. In addition to alerts, which can be processed in several different ways, the agent uses the management module to report operating statistics to the management
system. These statistics can then be centrally collated and summarized to analyze the performance of the different client applications.

- Communication. The management module provides the agent with a high-level communications protocol, which the agent uses to interact with other active components in the management framework. This allows the auditing and alerting functionality of the agent to send events over the network.

At this moment, we have only defined support for a couple of management systems, but the open API allows third-party vendors to add support for the agent in their management software.

The basic management system will use a simple back-end server for communication. The back-end server can either be accessed using a standard file-sharing protocol or using a customized HTTP service. The file-sharing protocol is inherited from F-Agent and has proven to be highly portable in many different network environments. HTTP is a generic protocol that has several existing implementations available and does not require the more complicated semantics of file-sharing. Either way, the agent can both send and receive information in the form of files.

The agent also includes a limited SNMP agent, which can be used to interface with existing management systems without support for our solution. Due to the nonexistent security of the SNMP protocol itself, we will not allow direct modification of the agent configuration using SNMP. However, SNMP can be used to transfer authorization certificates from a central location to workstations, as the signatures on the certificates are proof of their authenticity. The non-confidential statistics collected by the agent can also be reported using the SNMP protocol, although there is no way to ensure the integrity of those statistics at the moment. The SNMP server simply consults the agent for specific OID values.

5.6 Auditing

The requirements for an auditing system as described in [Amo94] are based on experiences from several different auditing schemes. The purpose of auditing is to dissuade attackers that are aware of the auditing and to give administrators the possibility to trace past attacks that have occurred in spite of the auditing and security measures installed. The agent does not include auditing for operating system events that
are beyond its control, but contains the necessary auditing mechanisms to produce a reliable audit trail of its own activities and the activities under its control.

The auditing module is subject to policy management, as are all other agent modules. System policy dictates which of the implemented auditing points are deactivated at a given time, giving the administrator the possibility to control the amount and granularity of audit output from the workstation agents. We are, in a sense, describing negative authorization in the case of auditing. Disabling auditing is the operation that requires authorization – without proper authorization, all auditing points are active.

5.6.1 Current solutions

Quite often the concepts of auditing and logging are confused. Most modern systems include comprehensive logging facilities which allow applications to record events they consider important. The logs are often useful when an administrator has to trace activities on the system. In that sense they serve the purpose of an audit log. The review of system logs has proven to be an effective weapon against an attacker when the logs have been cross-referenced with logs of other systems. [Spa91]

Unfortunately, many current logs fail to meet some of the relevant requirements of an auditing system. The choice of events being logged is done in a rather arbitrary fashion according to “what seems to be useful”, often leaving out events that produce very large amounts of data. Even when the correct things are being logged, the logs are usually in a form that can be altered by the administrator. This means that an attacker that gains administrative privileges is able to remove any traces of the attack from the logs. This technique is becoming increasingly common in attacks that are launched against hosts on the Internet. There are also systems that send the log data over the network to a secure storage host, but if the data transmission itself is not adequately protected, a determined attacker can use network-level attacks to modify the logs.

The output of current logging systems are very often in a proprietary format, which means that correlating data between different systems is difficult or even impossible. There have been attempts to agree on a standard format for audit data, but so far they have not produced the desired impact on the base of installed systems. A heterogeneous environment with several different systems may be faced with a situation where there
may be enough data available to reconstruct an attack, but no way to extract that knowledge from the different logs.

5.6.2 Improved model

We have designed the agent with the shortcomings of current systems in mind. Implementing a full-fledged secure auditing system in the agent itself was considered, but the technical challenges that such a system presents dissuaded us from attempting it. We chose to address a few core shortcomings to achieve an improved model. The agent will improve the logging facilities on the systems where it is installed so that the result is something between a logging and an auditing system.

- Choosing the relevant items to audit. The agent audits all changes to the parts of the system that it manages. Audit function calls are placed on two levels in the agent. Calls to the agent's open APIs trigger at least one audit event, and several of the internal low-level function calls are audited separately. Thus, the audit log entries for an action includes both the external action and the possible internal activities that it triggers.

- Recording the correct information correctly. The agent uses the management module to output the auditing information. The agent's own audit information includes the entire authorization 5-tuple and the corresponding audit event. Thus, the audit trail includes not only what was attempted, but also what authorization was used for the attempt. The installed management modules may not understand all the information, but can extract the information they do know about for their own purposes. How the audit information is recorded depends on the installed modules and the policy controlling their use.

- Protecting the audit log. The agent's own management module cryptographically signs the audit event with a fast keyed hash. The key used for the signature is then hashed to produce the next signature key, and the original key is destroyed. The management server knows the initial secret signature key, and can reconstruct the entire signature chain. This approach protects the audit log from tampering after the fact, as the previous signature keys are known only to the audit server. The log entries can still be destroyed, but the tampering will not go unnoticed.
• Providing interoperability. The agent management modules can interface with the native logging facility on the system to produce auditing events from the system’s log. The information is incomplete from the agent’s point of view, but the audit log entries will still be in a standard format. As audit log events are collected from an entire network in the same format, intrusion detection systems can compare and collate the information independent of the proprietary formats of the different systems.

5.7 Alerting

Events requiring an alert can occur at any time. The agent may notice a series of attempts to gain unauthorized access or receive a “virus found” indication from one of the installed anti-virus scanners. Any of the audit events may be flagged as critical, which means that the agent raises an alert when an audit event occurs. It would seem that an audit event for a successful operation would rarely raise an alert, but we do not make a distinction based on this assumption. One can come up with a number of scenarios where the use of an audit success would cause an alert.

It could be argued that there is no difference between alerting and auditing, that a properly designed auditing schema would serve the same purpose. To a point, this is correct, but until we have a reliable real-time intrusion detection system based on a reliable auditing schema, there is a clear difference between the two in practice. There are no widely deployed intrusion detection systems that are able to deduce the need for alerts from the audit trail within the required performance parameters.

5.7.1 Requirements for alerts

The basic requirements for alerts are the same as for audit events. The mechanisms must record the relevant events with minimal impact on system performance. The alerts must contain the necessary information and be resistant to tampering. However, two systems with exactly the same auditing requirements may have different alerting policies.

The primary requirement for alerts is that they must be configurable, based on experience. The system policy will reflect the administrators’ knowledge of what kind of alerts are required and what kinds are undesirable. This knowledge is part of the
general security policy of the entire network. The importance of this factor can be illustrated with two examples.

The notorious accident at the Three Mile Island nuclear power plant proved just how critical an alerting configuration can be. Reviews of the audit logs have shown that the operators had all the information needed to prevent the accident. However, the initial malfunction led to a chain reaction that triggered an enormous number of alerts. The operators were unable to separate the relevant alerts from the others and thus made some critical errors that caused them to lose control over the reactor.

Macro viruses spread with documents trigger alerts when the documents are opened at a workstation. With the increased popularity of documents sent as email attachments, a single infected document sent to a large group of people may trigger virus alerts on hundreds or even thousands of workstations. The administrator will receive an alert for each of the separate infections, but will have a very hard time tracing the original document to permanently remove the infection. One alert is very much like the next one.

With a properly configured alerting policy, the administrator can tell the workstation agents to send their alerts to a management server. This server can then cross-reference the incoming alerts and send the administrator one single alert instead of a thousand.

With heterogeneous, distributed networks, alerting becomes more difficult. There is no guarantee that the workstation can access the management server at all times, as it may be a laptop PC on route to a location on the other side of the world. The alerting mechanism must also meet the requirements for off-line systems, and the alerting policy must be able to cater for different eventualities. An alert sent by a laptop using GSM SMS can reach the administrator independent of the location of the laptop itself.

The management module API of the agent contains a section for alerting-method plug-ins. When an alert is raised, the agent knows the current state of the workstation and can use the appropriate plug-ins. There are a large number of possible alerting methods.

- SNMP TRAPs
- Pagers
- GSM SMS messages
- SMTP and other email message systems
- Voice-mail voice synthesis messages
- Third-party management system alerts

5.7.2 Alert classes

There are several taxonomies for audit events and alerts, but from the agent’s point of view, the alerts can be divided into a few basic classes.

- Critical alerts affect the entire operation of the workstation. A kernel exception may have been raised or a critical subsystem, such as the hard disk controller, may have failed. This alert class is the most important, but the most difficult to serve reliably. When the agent detects the need for a critical alert, the system may only have a few milliseconds left before total failure, and many peripheral systems may already be inaccessible. The kernel-mode driver will make a best-effort attempt at serving the alert directly, using a loaded critical alert method, but there is no guarantee of delivery.

- Failure alerts require administrator intervention. The agent has detected that some part of itself or of the underlying system has permanently failed. This alert class can be served using the installed alert methods, but it requires a high priority on all systems. The failure alerts indicate some unrecoverable permanent error and can also be issued by the agent’s client applications.

- Malfunction alerts may require administrator intervention at a later time. The agent issues a malfunction alert when some subsystem temporarily does not respond as expected. The agent may try to restart or reload the subsystem after issuing the malfunction alert. If the subsystem still does not function correctly, the agent will send a failure alert. Client applications are at liberty to do the same with their own subsystems.

- Application alerts are defined by the applications. The agent only forwards the alerts as directed by policy, which may even include completely ignoring some of the alerts occasionally. Typical application alerts are alerts triggered when anti-virus software...
finds a virus; when encryption software detects attempts to decrypt files with the wrong key or passphrase; and similar events that the client applications determine to be more critical than simple audit events.

5.8 Hardware abstraction

The agent serves as an abstraction layer to its client applications. Internally, it also contains several abstraction layers that reduce the complexity of the clients. When a new method is added to the agent, it is instantly available to all the client applications, provided that the system policy allows it. The client applications only need to know one API, that of the agent, to access all the services supplied by the agent. Previously, each and every client application would have had to separately implement support for the different methods.

Accessing the services through the agent does impact slightly on the performance of the clients, but generally the overhead is compensated for by significantly improved versatility. The major exception to this rule are the encryption libraries, which need every bit of extra performance they can get. Thus, the agent does provide an interface to encryption hardware, but no abstraction for the software implementation. The client applications are expected to link with their own optimized cryptographic libraries for optimal software encryption performance.

5.8.1 Identity storage

We expect a large number of hardware identity tokens to be introduced during the next few years. The idea of storing digital signature keys on tamper-proof hardware tokens is not new, but only recently has the technology become cost-efficient enough to be available for large implementations. The authentication agent contains an open API for hardware tokens, which means that once a token interface is installed, all the applications can use it.

The agent authentication cache implements a software equivalent of the identity storage devices. Signature keys can be added to the cache and then used to sign authentication challenges as though the keys were residing on separate hardware. The agent will not allow the extraction of signature keys from the cache, and the keys are always stored on disk in encrypted form.
The main requirements for identity storage is that the identity cannot be accessed by anybody but its rightful keyholder and that any challenges presented for signing are known. The authentication module will forward challenges for signing only when the source of the challenge is properly authenticated.

5.8.2 Physical identification

The authentication method plug-ins can also be implemented in the form of physical identification devices. Some of the devices are actually capable of digital signatures, whereas others only provide an authentication token. Either way, the plug-in API abstracts the details of physical identification and allows the agent to interact with them regardless of their properties.

Biometric identification devices with very reliable performance are now available for a reasonable price. The use of these devices in authentication, without accounting for their unique properties, can have unexpected consequences. The False Acceptance Rate (FAR) of a biometric device indicates how probable an incorrect identification is; and it may be as high as 1/1,000. However, FARs of less than 1/1,000,000 are available with most devices. When using a biometric identification device for authentication, the risk caused by the FAR depends on what authorizations the authentication gives.

The clients may specify a FAR they consider acceptable, and only devices with a smaller FAR will be used in the authentication process. If a FAR of zero is specified, the agent will allow any of the biometric devices to be used. System policy may also control the acceptable authentication methods for different services and may specify a maximum FAR for critical services.

The introduction of biometric identification also has a dark side. If allowed, clients can use the biometric device to strongly identify the actual keyholder that is using the workstation at a given moment. With a FAR of 1/1,000,000, this is almost positive proof of the real identity of the user. The ability to reliably identify the user is something that can be abused, and should be tightly controlled by the security policy.

5.8.3 Encryption hardware

Software encryption performance varies depending on the algorithm used and of course on the processing power of the computer. Symmetric algorithms, such as IDEA and
Blowfish, encrypt between 10 and 20 Mbit/s on a modern Pentium-based PC. Asymmetric algorithms, such as RSA, are considerably slower. Generating a digital signature using a 2048-bit RSA key may take a second or two, which causes problems for protocol implementers.

The performance of asymmetric algorithms is rarely an issue, as the keys are only used to encrypt symmetric session keys, but the private key must be kept absolutely secure. The most secure approach is to embed the key in tamper-proof hardware that can also create digital signatures with the key. There is no global standard for such hardware, and we do not expect one to emerge, so the agent API creates an abstraction for digital signature devices. This allows integration of current and future solutions without changes in the client applications.

Software encryption using symmetric algorithms can be done at a reasonable rate on a single workstation with moderate processor load. Performance becomes an issue when real-time encryption over a network is required. Network connections of up to 4 Mbit/s can be served using software encryption as long as the processor is not loaded with other tasks. With modern networks achieving speeds over 100 Mbit/s, the encryption becomes a bottleneck on workstations, and servers that have to serve multiple encrypted connections will see drastic performance reduction. The only solution to the problem is to do the encryption with dedicated hardware.

The agent provides an abstraction for the peculiarities of the different hardware devices, giving the client applications a standard API for using encryption hardware. The abstraction level is lower here than in the other APIs, as the client applications must be aware of the option to be able to use it. The reason for this is two-fold. Applications that do not need the encryption hardware can use their own linked libraries, freeing the valuable hardware device for applications that really need it. Applications that need the highest performance version do not always have to pass through an abstraction that causes overhead; if a hardware implementation is not available, the applications can use their internal, optimized versions.

### 5.9 Public key infrastructure

The entire authorization concept presented by the agent depends on the proper use of public keys. The authorization certificates are signed. The software installation packages are signed. Users have to sign authorization challenges. And all this must be done in a
reliable manner. The agent provides secure storage for the private keys, but it also provides the client applications and the users the necessary public key infrastructure (PKI).

5.9.1 What is a PKI?

The general definition of a PKI is very broad. In simple terms, it is the means and methods for ensuring the validity of a given public key and for retrieving other valid public keys. However, the definition of “valid” in this context is a matter of interpretation. This necessarily complicates the agent implementation.

PGP considers a public key valid if the key is signed by a given number of known valid keys. This definition can be applied recursively to a certain depth, and the weight of the different keys can be adjusted by the user. This model is usually referred to as a “web of trust”, as it creates a web that emanates from the user’s own private key. The agent does not currently support PGP keys. Adding support for such keys has been considered.

X.509 considers a public key valid if the key is certified by a known Certification Authority (CA). A CA is considered to be known if it is the Root CA or certified by a known CA. This creates a tree of CAs and a “chain of trust” from the Root CA to every public key.

SPKI does not directly address the issue of key validity. When a verifier receives an authorization certificate chain, the certificates themselves contain enough information about the public keys used and where to obtain them if necessary. The verifier performs a 5-tuple reduction to reduce the certificate chain. If the chain was complete and the original issuer was the verifier, the chain is an authorization loop and finally reduces to a 5-tuple issued by the verifier.

The concept of “trust” in public keys is frequently misused and misunderstood. We prefer not to refer to a public key as being “trusted”, but rather consider a key to be either valid or invalid in regard to a certain authorization. Currently, we do not consider public keys to convey any sense of “trust” whatsoever, as the standards and practices do not cover the matter of trust.
5.9.2 Obtaining a public key

The agent supports retrieving keys for SPKI using the standard protocols so that 5-tuple reduction can be performed correctly. This functionality is embedded in the SPKI engine itself and is not directly available to the client applications. As the agent performs the authorization procedures, the client applications do not need to know how SPKI functions to be able to support it. The SPKI retrieval method is based on HTTP, which the agent implements as part of its management module.

The X.509 key support is quite different. To use an X.509 certificate for encryption, it must be obtained somewhere. The agent implements the LDAP directory access protocol, so that the user can browse X.500 directories for the required keys. Once found, a key can be stored in the user’s own X.509 address book for later use so that a lookup is not required every time a key is needed. X.509 keys can also be obtained using other methods, such as email attachments and WWW downloads. The agent can import any such keys that are stored in the standard X.509 file format.

The agent can use X.509 keys as part of the SPKI reduction chain as long as the keys are identified as X.509 keys. The keys are checked for validity before use, but no other semantics are attached to X.509 keys by the agent. If a key is not valid, the agent will act as though the correct key could not be retrieved.

5.9.3 Using a public key

Once obtained, a public key can be used to encrypt messages intended for the keyholder of the public key and to verify signatures made by that keyholder. When the agent is using a public key for authorization certification, the use is encoded in the certificates, but X.509 certificates do not contain the same information. As public key infrastructures do not contain enough information for the agent to interpret the intended use of a key, the final decision on the trustworthiness of a particular key falls on the user.

SPKI certificates encode the intended authorization in the certificates, so the agent can present the user with the certificate before applying it. The exact interpretation of the authorization may still be unclear, as the verifier is the final authority on what access an authorization certificate grants. The SPKI certificate format and authorization types have yet to find their final form, which means that the interpretation of the authorization fields may change slightly with time.
It is an unfortunate fact that the agent currently has to rely on the user as the ultimate source of authority on the use of the public keys it handles. If the user misuses a key, either intentionally or unintentionally, the agent has little or no possibilities to detect and avoid the problem.
6. Conclusions

We have described a multi-purpose authorization and policy management agent. Our decision to make the policy manager the central decision-making module that all other modules must use allows us to implement a reference monitor for the agent. The reference monitor is able to strictly control the authorizations of the client applications that voluntarily delegate part of their functionality to the agent. The reference monitor does not apply to all system functionality and therefore it can be bypassed, but all access to the agent services is monitored.

Authorization certificates provide a finely granular way of expressing authorization and thereby system policy. The reduction of authorization certificate chains is a costly operation involving several public key operations, but the result of the reduction can internally be expressed as a capability. This close coupling between authorization certificates and capabilities lead us to believe that our system exhibits the same security properties as capability based systems. The use of authorization certificates removes the Achilles heel of capability based systems, the need to keep the capabilities secret at all times.

The solution we describe is feasible, as it generalizes several functions that are now usually implemented separately for each application. The agent provides services that we believe will eventually be closely integrated into the operating system itself. However, the trusted computing base of the agent we describe is very large, which suggests that operating system integration should be performed in the very same manner that we are using. If the entire operating system relies on the agent for the required functionality, the trusted computing base can be reduced to the few critical functions the agent requires to operate.

The use of policy-based management allows us to separate applications from their settings. This division makes the automatically administered installation of new applications easier, as the installation package contains no system-dependent components. All required installation parameters can be separately preset by the administrator to make the installation procedure completely non-interactive. This becomes especially valuable when application updates are distributed, as the update
created by the software vendor can be used without modification. The only requirement is that the necessary authorization and settings are available on the system.

The agent seamlessly integrates into existing and future management frameworks. The separate management module enables the agent to react dynamically to changes in its environment and allows the administrator to create general policies for different operational environments. With the rapid increase in mobile computing, the ability to instantly react to changes is becoming an essential part of any system and the security systems must reflect this fact.

We have named a number of open APIs with a high level of abstraction. This approach is similar to that of modern operating systems, which delegate the task of writing application dependent drivers to the vendors of the applications. The multiple open access points provide us with the opportunity to easily integrate new solutions as they appear, without sacrificing compatibility. We expect the APIs to evolve with time as new applications are found that do not fit the current abstraction. The APIs include mechanisms to ensure downward compatibility with earlier versions. The exact definition of the APIs will be an open process, where application designers will be allowed to give their input on what is required of the APIs.

Administering a network where all the workstations are running the policy-based authentication and management agent will be considerably easier than with most current solutions. Administrative procedures can be expressed as high-level policy decisions and are enforced automatically by the agents on the network. When necessary, the agents perform software updates and restrict user privileges. Throughout the process, a high level of security can be guaranteed, even though we recommend using a network level security protocol such as IPsec to further enhance the security of the network. When combined with IPsec, the agent provides a low-noise high-security framework based on authorization and policy management.

### 6.1 Suggestions for further study

The agent introduces several new critical points that can create new security threats. We have designed the agent framework with security as the key factor in all decisions, believing that a secure system can only be created if security is taken into account from the beginning. Nevertheless, a comprehensive risk analysis of the model should be done, and all the security features require open scrutiny by independent experts. It would be
naïve to assume that we are able to design a totally secure system right away when more experienced designers have failed.

The set of required authorizations still puzzles us. The concept of authorization certificates is new, even though capability-based systems have been available for some time. There is a need for further research aimed at creating a comprehensive taxonomy of authorizations and capabilities in a system based on authorization certificates. The concept of delegation of authority also requires close scrutiny, as the risk of violating the principle of least privilege is high. Determining the minimal set of authorizations for a certain task is surprisingly difficult, as the set of authorizations for a given system is a matter of policy.

The agent described here is designed for Windows NT 4.0. We have tried to keep much of the structure independent of the operating system, but achieving security does require close cooperation with the operating system internals. Further research is needed to isolate the operating-system dependent parts and to create an abstraction that hides the particularities of a given system. We are not even convinced that such an abstraction is possible, even though most current operating systems have many features in common.
7. Summary

We have described the legacy application F-Agent, the solutions it provides, and the challenges it currently faces. These challenges are the basis of this thesis, as a new solution is required within the near future. The current solution is reaching the limits of its scalability and does not meet all the security requirements of a modern environment. It uses proprietary solutions which do work as expected, but provide little or no interoperability with other systems.

We define a set of base requirements for the new solution from the perspective of a global internetworked environment, where local area networks are joined to form global virtual private networks. The environment is heterogeneous and can be highly dynamic, which places new requirements on performance, interoperability, and the capability to adapt to changes.

The requirements include a clear, modular structure with well-defined inter-module interfaces. This structure is intended to provide the necessary abstraction to provide interoperability with new solutions. Some of the interfaces are defined as open APIs, which allows third-party solutions to be easily integrated into the framework.

The new solution will securely handle software management, authorization, authentication, identity management, auditing, and necessary cryptography, all under the supervision of a policy-based management module. The policy manager allows users to act in both the roles of users and administrators according to their authorization certificates.

We present some of the main concepts that are relevant to policy-based management and authorization certification. The definitions of "identity", "authentication", "authorization", and "trust" all affect our concept of "policy". The definitions are not universal, but adapt well to our model of SPKI authorization certificates describing system policy. As the definitions evolve, so will the interpretation of our work.

We describe the structure of a multi-purpose authorization and policy management agent. The major components together form a policy-based management framework for their client applications. User authentication and authorization is handled by a separate authorization module, which provide the other modules with the necessary authorizations in the form of capabilities. Automated, remotely administered software
updates can be scheduled to be performed securely over a network, with the configuration management being separately handled by the policy manager.

The built-in reference monitor includes audit points and alerting functionality. The audit trail is protected from tampering after the fact to ensure that any changes can be detected. The audit data is presented in a standard format to the management system to allow information from different systems to be easily compared. Administrators can control the alerting functionality of the agent to ensure that the excess alerting is avoided and critical alerts are retained.

The agent provides abstraction for both hardware and public key operations. The abstraction level depends on the required functionality and client applications need to be aware of how to correctly apply cryptography. The agent is able to interpret the semantics of authorization certificates, but the ultimate trusted source for interpreting a certificate is still the user. Current certificate formats do not provide enough information for the agent to reliably determine the trustworthiness of certificate chains.

The properties of the agent solution are discussed and suggestions for further study are presented.
References


<URL:http://www.mcg.org.br/trustprop.txt>

<URL:http://www.cis.upenn.edu/~KeyKOS/ConfusedDeputy.html>

<URL:http://www.cis.upenn.edu/~KeyKOS/Confinement.html>

<URL:http://www.cis.upenn.edu/~KeyKOS/Security.html>


Appendix A: Open APIs

At the time of writing, the F-Secure CounterSign™ Architecture consists of the following APIs. The agent described in this thesis requires considerable additions to the existing APIs.

**F-Secure Gatekeeper Device Driver API**

- Architecture support for kernel-mode plug-in device drivers
- Simplified interface to file system events, regardless of file system type

**F-Secure Manager Extension API**

- Architecture support for user-mode plug-in drivers
- Interface for third-party management modules

**F-Secure File Management Module API**

- Centralized file management functions
- Support for archive formats and compound file formats

**F-Secure Anti-Virus Plug-In API**

- Architecture support for anti-virus plug-in extensions

**F-Secure Anti-Virus Network Protocol API**

- Abstract network protocol definition
- Future extensions will provide management modules with a network-independent protocol