Contract Based Late Security Binding

Ioannis Sakarelis, Thomas Strang, Thaddaeus Dorsch, Patrick Robertson

Institute for Communications and Navigation, German Aerospace Center (DLR)
P.O. Box 1116, D-82230 Wessling, Germany
E-mail: {firstname.lastname}@dlr.de

Abstract

In this paper we describe a security architecture, that allows emerging computation platforms such as PDA's, set-top boxes and mobile phones to host a variety of applications in a secure fashion. We will introduce a framework to distribute applications and associated security modules - so called security bodies - between the application developer, a trust center and the user's platform. We extend on currently known security frameworks and thereby introduce greater flexibility in the level of security and safety. Specifically, we define a security body associated with a signed application. This security body contains the public key for the application, as well as rules and software plug-ins governing the behaviour of the application at runtime. The active elements of the security body can take into account the current status at the end-user device, which may not be known in advance. We explore two procedures of interaction between a trust center and an application developer, the first one allowing a less restrictive certification procedure of applications. The second one gives the trust center direct control over signing each application release and also lets the trust center to validate applications in advance. A key feature is the fact, that application and security bodies, although they belong together, may be distributed separately. One application might even have several security bodies for different contexts. An important consequence for the end-user is that for each application he is provided with a trustworthy security configuration by default.

1. Introduction

Wireless Internet, mobile multimedia, integrated information services are some keywords that describe the demands our information society. Major R&D activities are focused on open software architectures that will enable the users to select from a bouquet of applications and data conveyed over different network topologies and services. The user will be unaware of crossing boundaries of different networks and will be unaware of which network is actually used, i.e. the information will reach the final user through the most adequate network depending on the service necessities [1].

Open software platforms will be based on open standards and will create a horizontal market for service, content providers and equipment manufactures. Everyone will be able to develop applications. Our vision is that also everyone will be able to provide such applications and services to a platform in the most easy way.

Enhanced requirements on security and safety are becoming rapidly more important. It is essential that applications will run in a proper and correct manner. In a world accessible by

everyone it will not be possible to check every application concerning their security and safety aspects. Therefore, the safety and security requirements placed upon the computational environment where the code will be executed must be very strict. The success of such open software platforms will greatly based on the aspects of security and safety.

In this context *Security* is defined as the concept of protecting privacy, integrity and availability in the face of malicious attacks, whereas *Safety* can be described as reducing the risk of mistakes and or unintended behaviour.

The computational environment may be a desktop computer, an PDA, in-car infotainment platform, a set-top box or a mobile phone. Assuming the computational platform (CP) is equipped with Java running on a Virtual machine (VM), far-reaching security features are already built in (Security Manager, Access Controller, policy files etc.).

In an open framework applications may misuse the computational platform in various ways. Some of the security problems that may occur are the following [2]:

- **Integrity**: An application can destroy or change the resources or services of a computational platform by reconfiguring, modifying or erasing them out of memory. Also an application can attack other applications sharing the same CP.
- **Availability**: An application may overload a resource or service due to constantly consuming network connections or using a great portion of CPU cycles available. Under these circumstances other applications may not be executed.
- **Secrecy**: An application may access and steal private information from the CP.

In contrast, the following safety aspects arise concerning the behaviour of an application:

- **Context**: Under certain circumstances an application has to respect the specific runtime context. An advertisement application cannot popup while I am reading important news.
- **Prediction**: An application has to run exactly how it was designed.

In order to provide security and safety when an application arrives at an open software platform the system must:

1. Check and accept the authenticity of the credentials of the application.
2. Identify the application and/or service provider
3. Authorize access to appropriate resources based on the identifications and credentials
4. Allow execution based on the authorizations and security/safety policy,
5. Monitor and control access to system throughout the execution.

Our approach meets these requirements in a flexible and extendable way, providing platform openness as well as platform control.

2. Conventional approach

An excellent paradigm for mobile code that is delivered through networks to the user's computational platform are Java applets. How are the security related issues solved in this case? In general Java applets are divided in the categories *unsigned* and *signed* [2]. Applets which are treated as unsigned will be executed in a "sandbox". The "sandbox" model provides a very restricted environment in which unsigned code obtained from the open network can run. Trusted applets are "signed" applets and are delivered with their respective signatures in signed JAR (Java ARchive) files. Trusted applets may be subject to a security policy. The security policy defines the set of permissions available for code from various signers or locations and can be configured by a user or a system administrator. Each permission specifies a permitted access to

a particular resource, such as read and write access to a specified file or directory or connect access to a given host and port.

There is a very well known procedure which allows one to sign Java applets as trusted and to export an certificate which identifies this applet in a unique way. However, a certificate is more trustworthy if it is signed by a certification authority (CA). To get a certificate signed by a CA the application developer (AD) sends a certificate signing request (CSR) to a CA. The CA will authenticate the AD, sign and return a certificate authenticating the public key. That is, the CA vouches that the AD is the owner of the public key by signing the certificate (see Figure 1).

The downloaded applet is subsequently treated as trusted because the owner of the computational platform believes the CA that identifies the AD. The acceptance of a certificate does not imply any trust to the application's behaviour, it only identifies the AD. Moreover, there is no assurance that the application will behave in a proper and correct manner.

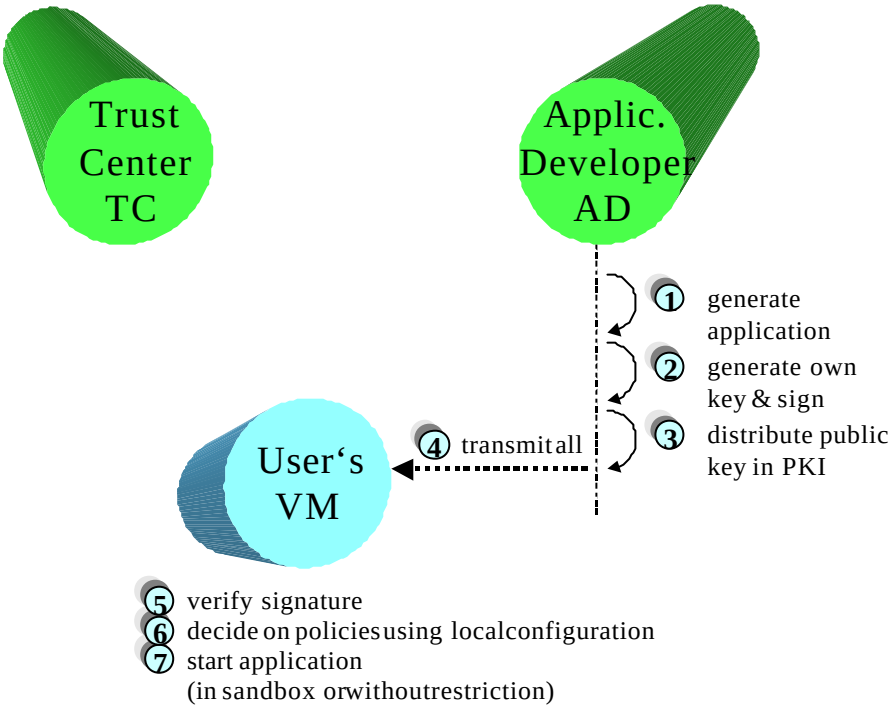

Figure 1: Java Signing Procedure

Another example for distributing applications over networks to users platforms is the Multimedia Home Platform (MHP) [3]. The MHP is an open Digital Video Broadcast (DVB) [4] standard for delivering applications via DVB to the users home set-top box. In MHP version 1.0 the applications are, concerning the security aspects, analogue to the above described Java security model for applets, separated to signed and unsigned. MHP applications are `java.Xlets` [5] and have a lifecycle mechanism like applets. In MHP the signed applications can be tied to a Permission Request File (PRF). The PRF is an XML file that describes the permissions that will be granted to the signed application. The PRF is mapped at runtime to Java Permission Objects. Unsigned applications and signed applications without a PRF have access to all the API's for which there is no permission signalling defined and will run in a sandbox (see Figure 2).

Suppose that someone wants to write an enhanced MHP application that makes extensive usage of security related API's. What has he to do in order to provide such an application and gain

access to protected by permissions API's? The answer is that he has to convince the broadcaster/service provider who holds the access medium to the platform about the security and safety of his application. The broadcaster/service provider probably will test this application concerning security and safety and will after successful testing provide the necessary certificate and sign the application as well as the PRF for this application and finally distribute this application through any access medium.

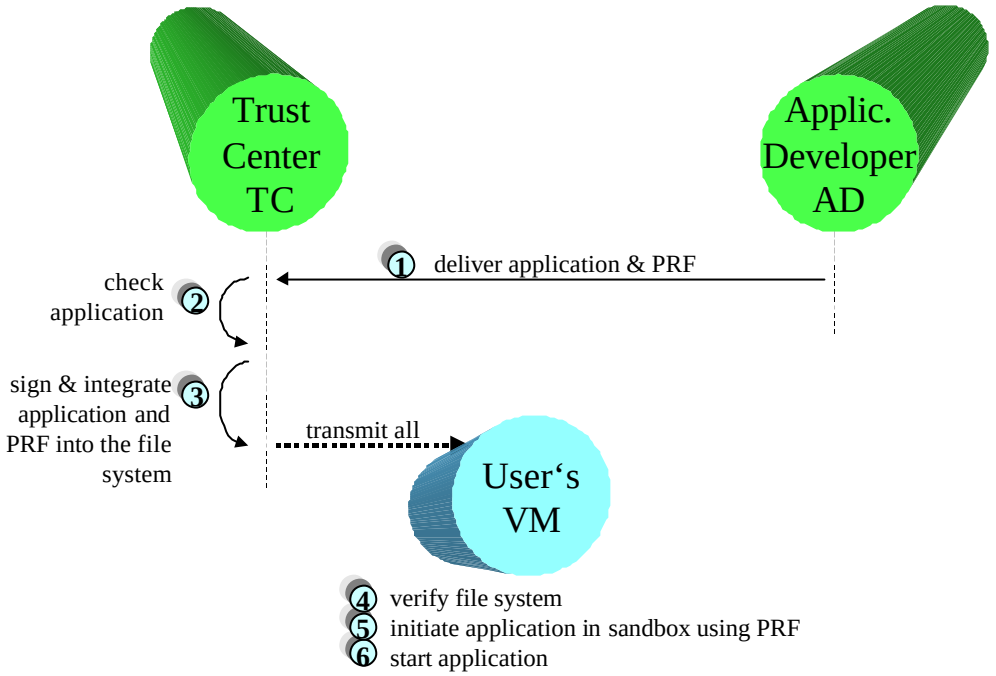

Figure 2: MHP Certification Procedure

The drawbacks of the paradigms used today are

- the user has blindly to trust an application (applet) by accepting the certificate
- the application is well proofed and tested by an authority before it is released and distributed to the platform

Moreover, none of the security features implemented today are designed with the perspective of the vendor of an open platform in mind. A truly open platform means that everyone is allowed and able to offer services for the platform. But none of the already implemented security features takes the vendor into account who wants to open his platform and have a fine-grained access control to its resources simultaneously.

Application developers do not know about the restrictions activated on the user's side. So it is very difficult and error-prone for the developer to take all the different situations into account.

The approach introduced in the next chapter aims to

- release the user from the responsibility of accepting a certificate
- allow the CA to define the application behaviour depending on a specific context at a CP
- if desired, release the CA from testing an application
- if desired, require that the CA be involved in certifying each application release

Our work answers the question: "How can I provide applications in an open software platform where everyone has access, in a secure, safety, easy, flexible and extendable manner?"

3. The New Approach

As mentioned above, Java has already far-reaching security features tied into the virtual machine (VM). When starting a Java application in a VM, various security mechanisms like the SecurityManager, AccessController, policy files and ClassLoaders work together to provide the Java VM as a secure, ready-built platform on which to allow the applications to run in a secure fashion.

Most of them are designed with the perspective of the user in mind, which means the user should be able and in duty to give or deny the application access rights by defining policy rules. In fact, writing such policy rules is not very easy. Thus, most of the users grant all permissions to avoid annoying warnings and access violation messages to their applications.

The basic idea of our approach is to solve these problems by binding a *security & safety context* to the application, which is called a *security body.* The security body is issued by a trust center, which specifies a scope and the access rights to an application, which has been previously negotiated between the trust center (TC) and the application developer (AD). Thus, we have three parties involved (see Figure 3).

One of the most interesting and innovative aspects of this approach is the negotiation process between the AD and the TC. Whereas a trust center usually certifies the *identity* of someone or something, the TC in this scenario goes one step further: The TC certifies also the *permissions* an application will be granted. The certification is preceded by a negotiation process between the AD and the TC, where the AD requests the desired permissions for the application to develop (e.g. to have the permission to access a specific sensor resource out of the users virtual machine). This security model allows the trust center to define the framework any application runs in concerning legal issues, priority, integrity, privacy etc. in the users VM. The negotiation procedure enables a accurately defined and comprehensible instrument of control, where in case of contempt penalty clauses can be deployed. Once the set of permissions are defined and certified, the application developer can assume to have a well defined security environment for the application.

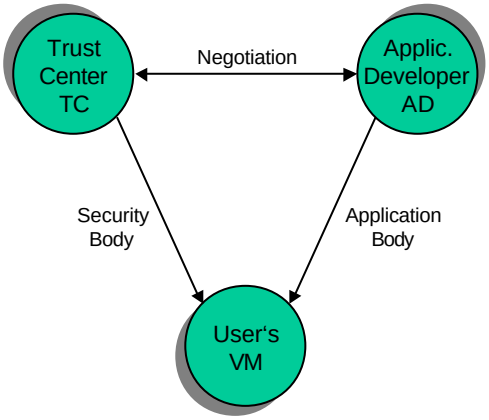

Figure 3: Party-of-three model

This party-of-three strategy requires an interaction and distribution model between the participating entities. We defined two different procedures of certification:

- Less Restrictive Certification Procedure
- More Restrictive Certification Procedure

which differ mainly in the level of security guarantees. We will explain them both in the next sections.

3.1 Less Restrictive Certification Procedure

Our less restrictive or basic certification procedure negotiates and establishes a legal contract between the application developer (or service provider) and the trust center. The application developer generates an key pair whose public key is registered with the trust center. After implementing an application regarding the negotiated application access rights and policy rules, the developer signs the application using the private key. The application and the resulting signature build together the *application body*, which is distributed in any suitable manner to the user. In parallel the trust center generates a list of application access rights, and signs the public applications key and the list together with the trust center's private key, building the security body, which is distributed to the user independently or attached to the application body. For details, have a look at Figure 4.

Before the application is started in the user's virtual machine, the security body's signature is validated with the well known public key of the trust center. When valid, the public application key contained inside the security body is used to identify the author of the application and to validate the application body's signature. Only when this check is positive, the application is started by making use of the access rights contained within the security body.

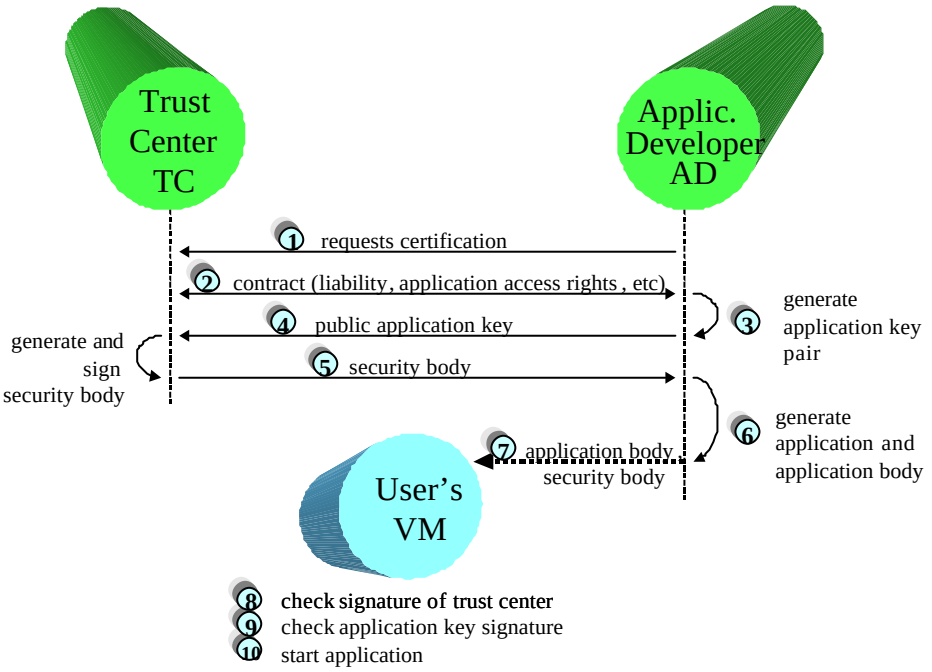

Figure 4: Less Restrictive Certification Procedure

This basically guarantees from the user's point of view, that

a) the generator of the key pair is the author of the application, because only he knows the secret application key.
b) the validity of the security body can be checked by using the well-known public key of the trust center. That means no one else than the trust center, especially not the application developer, is able to create a valid security body containing the AD's public key and the list of access rights negotiated between the TC and the AD.

c) the permissions negotiated related to the specific application (which is identified by the public application key) between the TC and the AD are taken into account.

One problem in this basic procedure is the possibility of misuse of a certificate on the application developer's side. For instance an AD could use an application key registered with the TC for a security body with far-reaching access rights to sign another application which has not been negotiated about with the TC previously.

Further on, the basic procedure does not guarantee a proper functionality of the application itself, because there is no checking instance for the application. Both can be partly improved by using the enhanced certification procedure, which is introduced in the next section.

3.2 More Restrictive Certification Procedure

The more restrictive or enhanced certification procedure is structured very similar to the basic one. The main difference is where the application key pair is generated: Whereas the AD creates the keys in the basic procedure, the TC is responsible for generating the application key pair and signing the application body in the enhanced procedure (see Figure 5).

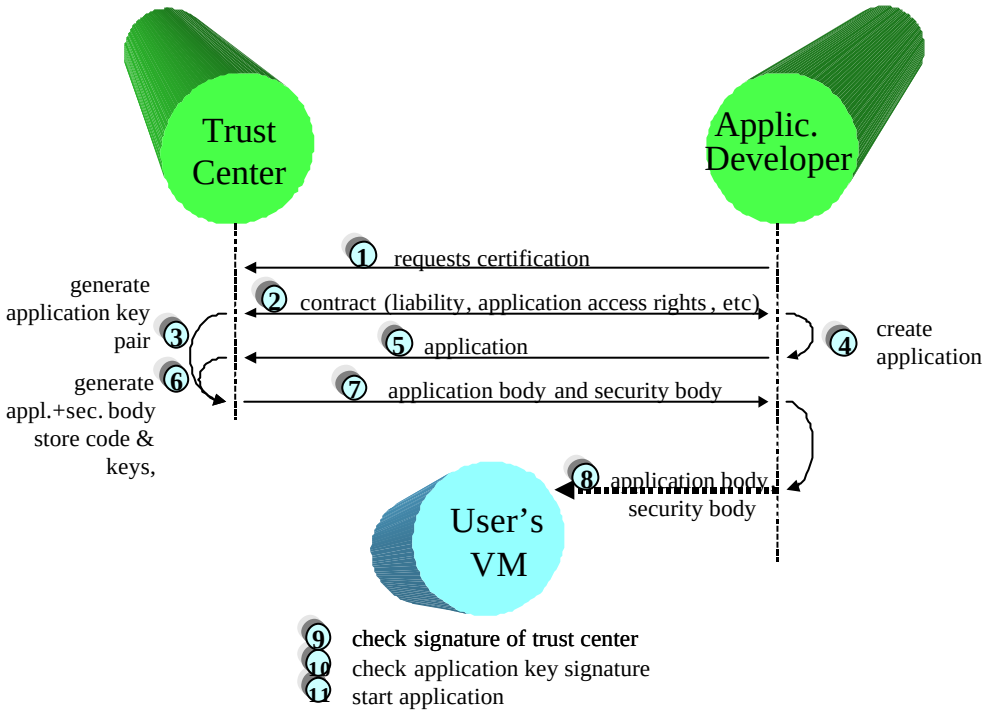

Figure 5: More Restrictive Certification Procedure

This gives the trust center the opportunity of checking the application before signing it. This can be done in different levels of detail. For the highest security requirements this could be done by examine the application's source code, and for less restrictive requirements this could be done by exercising some spot tests with random or critical input values in different runtime states.

When deploying the enhanced procedure, even the application developer is not able to distribute an abused application with a valid signature, because the AD does not know the private application key.

3.3 Application and Security Body

The presented certification procedures base on a pair of information units: the application body and the security body (see Figure 6).

The application body consists of the classes of an application as well as all associated resource files, typically combined in a JAR file, and is signed by using the private application key. This key is generated and stored either by the application developer (Less Restrictive Certification Procedure) or the trust center (More Restrictive Certification Procedure).

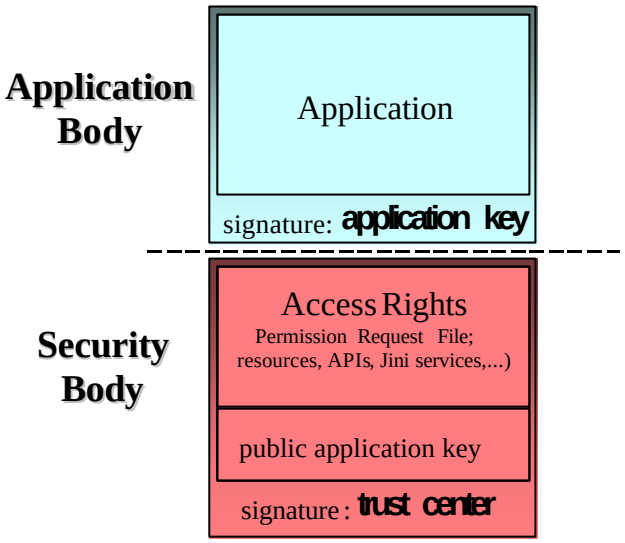

Figure 6: Application and Security Body - Overview

The security body is derived from the principles of a X.509 certificate [6]. The following ASN.1 data type represents X.509v3 certificates (see Figure 7):

```
  Certificate       ::=    SIGNED{ SEQUENCE{
    version                    [0] Version DEFAULT v1,
    serialNumber                   CertificateSerialNumber,
    signature                      AlgorithmIdentifier,
    issuer                         Name,
    validity                       Validity,
    subject                        Name,
    subjectPublicKeyInfo           SubjectPublicKeyInfo,
    issuerUniqueIdentifier     [1] IMPLICIT UniqueIdentifier OPTIONAL,
                                   -- if present, version must be v2 or v3
    subjectUniqueIdentifier    [2] IMPLICIT UniqueIdentifier OPTIONAL,
                                   -- if present, version must be v2 or v3
    extensions                 [3] Extensions OPTIONAL
                                   -- if present, version must be v3
    }}
  Extensions        ::=    SEQUENCE OF Extension
  Extension         ::=    SEQUENCE{
    extnId                         EXTENSION.&id({ExtensionSet}),
    critical                       BOOLEAN DEFAULT FALSE,
    extnValue                      OCTET STRING
                                   -- contains a DER encoding of a value of type
                                   --  &ExtnType  for  ext.  obj.  identified  by
extnId
    }}
```

Figure 7: ASN.1 representation of X.509v3 certificates

By deriving the security body from X.509 certificates we are able to re-use any established PKI, especially in but not restricted to TCP/IP networks [7].

The security body consists of a collection of access control tags which can have a very different characteristics. One may see this as a certified Java policy file, or as an expanded version of a permission request file used in the MHP project. The exact format hasn't be defined yet, but will be an implementation of the *extensions* field in the X.509 certificate. Proper elements of the collection will be

- a list of accessible API's
- allowed resources (eg. files, network connections)
- Jini service groups or single Jini services [8]
- Context information
- Priority rules

Any element is marked as exclude or include, thus the elements can be listed in a *positive list* and a *negative list*.

One of the main advantages of this approach is its scalability. The trust center may sign a security body granting sensitive access rights to the AD for one application, and may sign a security body with very restrictive access rights to the same AD for another application. The trust center may also sign a set of security bodies for common use, e.g. one security body for advertising applications, and another one for in-car sensor read access applications. Even multiple applications may share the same security body, if they are signed by the AD or TC with the same private application key. New certificates may be omitted very fast, what is necessary for the rapid application development nowadays. Also is the system extendable, which is important bearing in mind new access control technologies.

As described so far, the security body is a passive tool for security control. But likewise the trust center can certify a collection of access control tags, it can also certify some bytecode which extends the security mechanisms at the client side like a plug-in. This bytecode is executable, therefor we call a security body containing access control tags as well as byte code *active*, and due to its active controlling behaviour by following the access control rules we call the new component "Policeman" (see Figure 8).

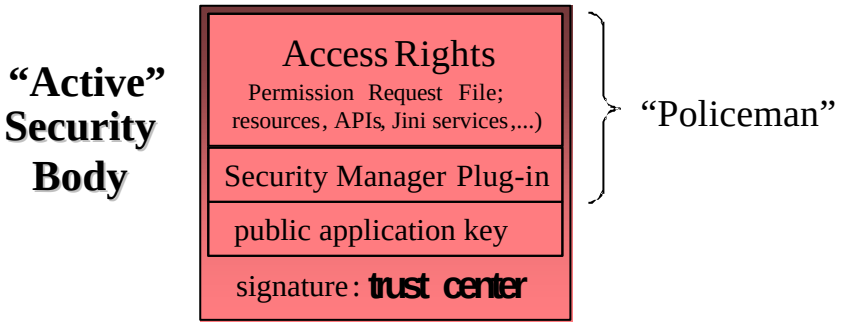

Figure 8: Active Security Body

The policeman becomes an active part of the user's virtual machine. Thus, he is not only able to extend functions of the VM's security manager or access controller in the *security* context. Moreover, it is possible to extend the virtual machines facilities for improving the *safety* context. This is basically done in the VM among others by the use of type-safety, the garbage collector or the bytecode verifier. Here the policeman gives the opportunity to extend those features with further checks in an application specific context.

A special policeman can either be negotiated between the application developer (or service provider) and the trust center, or one of a set of standard implementations provided by a trusted source may be elected by the trust center.

4. Conclusions

In this paper we have presented a novel security architecture for an open software platform that allows scalable levels of effective security within one framework. Examples for such systems are in-car computation platforms, platforms for handheld mobile devices, and others. The basis of our framework is to separate the interaction of application developers and trusted centers from the technical representation of necessary certificates. By adopting a two-tiered structure whereby an application body is separately represented from a security body, we allow degrees of freedom in the negotiation between application developers and trust centers, and in the distribution methods. Furthermore, our security body can be interpreted as a "policeman" carrying an active plug-in for a security manager on the platform, thereby allowing for even more flexibility in controlling the rights of an application. An example for the influence of the context of the terminal on the specific rights of an application is an in-car scenario, where a TV-viewing application's rights depend on the geographic location or the vehicle's speed.

In contrast to simple systems, where the decision on the security of an application lies in the domain of the end-user, our approach shifts this responsibility to one or more trust centers. Our approach can be employed in two major scenarios: the first is one in which the application developer requests and negotiates a set of rights for his application. Afterwards he is assigned a security body for use with his application. A legal contract is needed to ensure that the application provider does not misuse his assigned rights. With each new version of the particular application being released, the trust center need not be involved. This open approach would be conducive to many new applications being developed rapidly. In the second, more restrictive and cautious scenario, the trust center signs each application body with each new version. Common to both approaches is the ability to distribute the security body in a separate fashion from the application body: For example, the application may be broadcasted over a wireless network, and the security body might be provided on a smart card. Naturally, this carries the additional advantage that one application may have different security bodies; for example one different security body for each manufacturer of an in-car platform, without the need of distributing the application many times over. Another example is where two different security bodies govern the rights and behaviour of one and the same application when running on either a set-top box or on a handheld device.

Both the above approaches can be implemented at the same time, and the decision can be made from one application to the next: they have no implications whatsoever on the platform's behaviour. Our framework can be implemented using X.509 certificates and can thus rely on any PKI.

References

[1] Multimedia Car Platform (http://mcp.fantastic.ch)
[2] Li Gong, Inside Java 2 Platform Security, Addison-Wesley, 1999
[3] Multimedia Home Platform (http://mhp.org)
[4] Digital Video Broadcast (http://dvb.org)
[5] Java TV API (http://java.sun.com/products/javatv/)
[6] ITU-T Recommendation X.509, "IT – OSI – The Directory: Authentication Framework", 1997
[7] RFC2459, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile", Jan. 1999
[8] K. Arnold, B. O'Sullivan, R.W. Scheifler, J. Waldo, and A. Wollrath, The Jini Specification, Addison-Wesley, 1999