Institut EURECOM
Corporate Communications
2229, route des Crêtes BP 193
06904 Sophia Antipolis
(France)

Research Report[a] N^o 101 — RR-04-101

Protecting Applications and Devices in Nomadic Business Environments

Laurent Bussard and Yves Roudier

April 2, 2004

[a] This research was supported by European Union Project '*WiTness – Wireless Trust for Mobile Business*' [wit04] and by the Institut Eurécom.

Protecting Applications and Devices in Nomadic Business Environments

Laurent Bussard and Yves Roudier

Institut Eurecom
Corporate Communications
2229, route des Crêtes BP 193
06904 Sophia Antipolis
(France)
{bussard,roudier}@eurecom.fr

Abstract. This paper presents a pragmatic approach to protect the integrity of execution of an application in a nomadic business environment. Applications run in such contexts are based on the federation of appliances collaborating through direct communication as well as through the distribution of mobile pieces of code. Securing the operation of an application means protecting mobile code since execution environments may misbehave as well as protecting the environments because mobile pieces of code are potentially malicious. After reviewing several such protection techniques, an architecture for securing business-to-employee and business-to-business nomadic applications is drafted based on the trust model that can be assumed in such a context.

Keywords: mobile code protection, execution environment protection, device certification, attribute certificates, pervasive computing.

Introduction

Computer users are becoming more and more mobile thanks to the deployment of wireless technologies and to the increasing availability of mobile personal devices. Nomadic computing makes it possible for users to take advantage not only of his handheld or even wearable devices, but also of the appliances in his immediate vicinity, even if they do not belong to him. Enabling an application in such a system means accessing global and local communication infrastructures. For instance, UMTS can be used for communications with remote servers while Bluetooth will enable a pocket device to access surrounding appliances (e.g. printers, screens, sensors).

Nomadic application thus range from over the air access to a classical distributed service provided by a remote server to a set of mobile codes dispersed over close communicating devices, which is generally called a federation of devices. The latter organization helps alleviate the limitations of on-site available communication channels (i.e. restricted bandwidth, long round-trip time, or expensive cost) or the limitations of mobile devices (i.e. lack of computational power, screen size). For instance, a user traveling with a cell-phone will much more efficiently edit a document with a local public terminal than on the keyboard and screen of his phone.

Nomadic computing is especially interesting for a mobile corporate workforce, like salesmen visiting their customers. In this context, security becomes a major concern. First, access to the corporate resources and data must be controlled. Second, the safety of the operations performed by a user depends in fact directly on the integrity of execution of a program on devices that will not, for most of them, be owned by the employee or his company, and that may potentially be malicious. This is for instance what happens when a public terminal is used to edit a document that is subsequently signed with the employee's cell-phone (assuming the employee's private key is held by his SIM card). To ensure the *what you see is what you sign* principle, it is necessary to verify the integrity of execution of the editor. Finally, it is necessary to protect public appliances offering some service from hostile users uploading some malicious mobile code in order to attack the environment hosting it. If not enforced, such appliances might be good candidates as Trojan horses of a new kind, unbeknownst to their owner.

Application protection and devices protection have often been discussed in the literature about mobile code security and have proven quite difficult to tackle [ST98,BV99,LBR02,NL98]. In contrast with these works, this paper, which extends a previous work on data protection [BRKC03], suggests that both issues be seen in terms of trust relationships:

– Can the terminal trust this piece of code and give it access to resources?
– Can the user trust this terminal to run some part of an application?

We propose a pragmatic way to evaluate the security-level of pieces of code and devices in the very specific context of business-to-employee (B2E) and business-to-business (B2B) nomadic applications. Access control as well as host and code protection can thus be defined jointly.

This paper is organized as follows: Section 1 first gives a description of existing approaches to protecting environment and code parts. Section 2 presents the architecture of the nomadic business environments we envision, including the security requirements pursued. Section 3 gives an overview of the proposed architecture and how to specify relationships between entities. Section 4 describes possible enhancement of the Java security mechanism in order to protect devices. Finally, Section 5 proposes a pragmatic approach to ensure the integrity and confidentiality of execution of an application distributed on some surrounding devices.

1 Approaches to Environment Protection and Code Protection

Nomadic computing requires distributing data and pieces of code in a federation of devices that are not always controlled by the user. The problem addressed in this paper is twofold: on one hand, attacks may be performed by mobile programs against the execution environment and its resources; on the other hand, mobile code and data may be subverted by a malicious execution environment. This section presents mechanisms dedicated to the former issue, which has been widely addressed [LMR00], mechanisms to deal with the latter issue, and some more global approaches. Data integrity and confidentiality has been more classically tackled through encryption and digital signature for instance.

1.1 Protecting Execution Environments

Protecting vital resources against potentially malicious pieces of code has been widely addressed in operating systems and virtual machines. This section lists several approaches and their relevance for securing nomadic B2E or B2B applications.

VM approaches These approaches address the protection of the environment through the isolation of the potentially malicious code.

Sandbox: The sandbox model is the original security model provided by Java. It offers a severely restricted environment (the sandbox) in which untrusted pieces of code are executed. Local code is trusted and has access to resources (e.g. file system, network) while downloaded code is untrusted and cannot leave the sandbox. This initial mechanism is still widely deployed: it is the default behavior of browsers (i.e. without java plug-in), it is also used in lightweight environments such as J2ME and Personal Java that run on cell-phones and PDAs. Finally, the applet firewall mechanism of Java cards has similar properties. This mechanism has now been superseded by the Java 2 security model.

Java 2 Security Model: The sandbox model has been enhanced with new security features [GMPS97]. There is no more built-in concept defining that local code is trusted and remote code untrusted but each piece of code receives different rights depending on its origin (i.e. URL), on the signature, and recently on the entity who runs the code. The access control to resources is fine-grained and easy to configure. Permissions allow the definition of rights and programmer can define application specific permissions (accessing a smart card, etc.). Security Policies are used to associate permissions to pieces of code. The work presented in this paper uses and extends those mechanisms.

JavaSeal: JavaSeal [BV99] proposes a security framework to ensure strong security between mobile agents. Confinement mechanism avoids covert channels between agents. Mediation ensures that security controls can be added between pieces of code. Finally, local denial of services attacks are avoided by finely controlling the resources (i.e. memory, computational power) used by agents. This offers interesting security properties that are out of our initial scope. However, JavaSeal could be combined with the approach proposed in this paper to offer a full featured platform for securing mobile code in pervasive computing.

Proof-carrying code An approach to host protection is to statically type-check the mobile code; the code is then run without any expensive runtime checks. Promising results were obtained in this area by the proof-carrying code (PCC) work [NL98]. In Proof-Carrying Code, the host first asks for proof that the code respects his security policy before he actually agrees to run it. The code owner sends the program and an accompanying proof, using a set of axioms and rewriting rules. After receiving the code, the host can then check the program with the guidance of the proof. This can be seen as a form of type checking of the program, since the proof is directly derived from it. In PCC, checking the proof is relatively simple compared to constructing it, thus this technique does not impose much computational burden on the execution environment. However, automating the proof generation is still an open problem.

Two security requirements specific to nomadic systems are not fulfilled by those approaches: a way to define rights of a piece of code in a distributed way that should make possible the delegation of rights between entities in charge of certifying pieces of code; and a mechanism to dynamically change the rights of an application is also necessary. Section 4 proposes an extension of the Java 2 security model dedicated to nomadic computing.

1.2 Protecting Mobile Codes

Protecting nomadic applications often requires protecting the mobile code parts that make it up. Protecting a mobile code against the environment that executes it is notoriously difficult. Verifying the environment trustworthiness is possible with some computer architectures. Other architectures in which this verification is impossible make it necessary to resort to techniques that render the understanding of the behavior of a piece of code extremely difficult in order to ensure its integrity or confidentiality of execution.

Protecting code with trusted platforms When the device that evaluates a piece of code is trustworthy, integrity and confidentiality of execution are ensured. Two approaches have been undertaken.

Neutral Tamper-Resistant Platform: A straightforward way to ensure that a device can be trusted is proposed by the Trusted Computing Group (TCG) [TCG04]. The hardware is tamper-resistant and certified. This hardware can verify whether a certified kernel is running on top of it. This kernel controls the OS, which can check applications. This architecture makes it possible to prove that a given environment is running. As long as all layers are trustworthy (i.e. certified and without implementation errors), it is possible to trust the environment. In other words, an application with some integrity or confidentiality requirements can be executed by any TCG public terminal with the guarantee that the host will not misbehave. For instance, it is possible to ensure that some confidential data will be erased when the user leaves the terminal, etc.

Trusted Tamper-Resistant Module: It is also possible to provide a trusted tamper-resistant hardware that will be in charge of executing applications. For instance operators provide SIM cards to their customers in order to have a piece of hardware that is totally under control. For obvious cost reasons, this approach suffers from limited performances. Moreover, it is not realistic to embed a personal hardware in all surrounding devices that can be involved. Finally, this approach only protects the execution of some program but does not protect inputs and outputs, e.g. keyboard and display of the cell-phone bearing the SIM card are still used.

Securing functions in malicious environments Protecting a function that is evaluated by a potentially malicious host is a first step towards application protection.

Secure function evaluation has been addressed by many researchers. Sander and Tschudin [ST98] defined a function hiding scheme and focused on non-interactive protocols. In their framework, the confidentiality of function $f()$ is assured by an encrypting transformation. The authors illustrated the concept with a method that allows computing with encrypted polynomials. The potentially malicious host evaluates the encrypted function and returns an encrypted result. [SYY99] and [LBR02] present non-interactive solutions for secure evaluation of Boolean circuits. Securing a program based on secure functions is not straightforward however, and may again require the use of a personal tamper-proof hardware.

Securing applications in malicious environments Securing the integrity and confidentiality of a whole application is difficult.

Integrity of Software Execution: Integrity of execution is the possibility for the program owner to verify the correctness of the execution. This problem has been extensively studied for achieving reliability (see for example [WB97] for a survey) but security requirements taking into account possible malicious behavior from the execution environment were not considered. Yee [Yee99] suggested the use of proof based techniques, in which the untrusted host has to forward a proof of the correctness of the execution together with the result. Complexity theory shows how to build proofs for NP-languages and how to build probabilistic checkable proofs (PCP) [AS98]. It requires checking only a subset of the proofs in order to assure the correctness of a statement.

Confidentiality of Software Execution: Malicious reverse engineering is an important problem of Java: byte code can easily be decompiled because it retains a large part of the original information and because applications based on powerful libraries are small. Obfuscation aims at transforming an application into one that is functionally identical to the original but that is much more difficult to understand. It is an empirical and mathematically unfounded solution (see [CTL96] for a catalogue of obfuscating transformations).

To summarize, on one hand, hardware solutions to protect pieces of code are difficult to deploy and expensive. Tamper-resistant modules are necessary to protect private keys but it is not always affordable to have a secure hardware that protects the execution of a whole application. Moreover, the process for certifying hardware is complex. On the other hand, there is no software solution to fully ensure integrity and/or confidentiality protection of a piece of code running on a malicious host. Indeed, all approaches presented in this section are restricted to a set of functions, are computationally expensive, and/or cannot be proven secure. Moreover, software manipulation does not ensure the protection of inputs and outputs.

1.3 Trust-Based Application Protection

Rather than focusing on mechanisms to tackle either the mobile code side or the environment side, this paper proposes a system wide and pragmatic mechanism common to both the protection of code and environment. Environment and code protection can be based on trust, i.e. authorizations and/or roles of application developers and security-level of runtime environments.

Approaches based on distributed policies for managing trust [KFP01,BFK99] do not take into account the security-level of execution environments. It is assumed that policies are always enforced and it is not possible to recognize an untrusted device from a trusted one. Policies are thus not sufficient for enforcing the protection of applications. We however envision policies to offer a flexible and high level specification of trust management.

Some works propose defining trust from scratch in a way similar to the creation of trust relationships among human beings. In this case, results of previous interactions [ENT$^+$02] or contextual information [SA02] can be necessary. In the business context described in this paper, trust is based on *a priori* knowledge. Recommendations, results of previous interactions, or even contextual information might further be used to extend this knowledge.

2 Nomadic Business Systems: A Security Architecture

We propose a framework for protecting the pieces of code, i.e. verifying the security-level of environment before allowing distribution, and protecting the environment, i.e. verifying that pieces of code are authorized do access resources, be they a database or a network connection.

2.1 Nomadic System Organization

Figure 1 shows how code distribution is done: different part of an application are tagged according to the security requirements and the security-level (SL) of each device is evaluated (see Section 5 and [BRKC03]). For instance, the signature related operation of an application has to be done in a trusted enough environment. Each piece of code receives short-term authorization to access resources (see Section 4). For instance, a word processor can call the signature function but a game cannot.

Servers, which are not managed by the user and whose trustworthiness may be questioned, may anyway have to deal with confidential data. Moreover, in

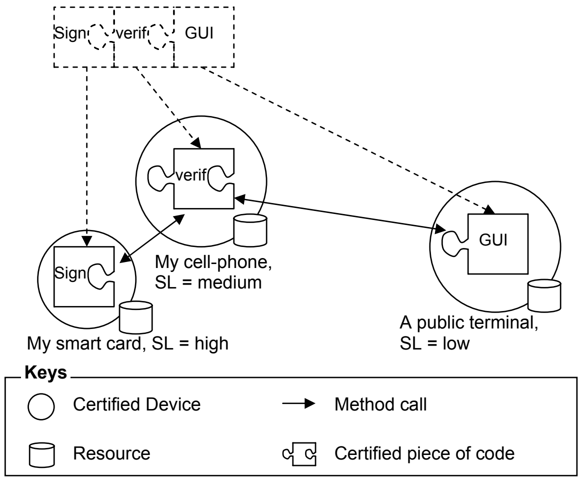

Fig. 1. General overview: certified pieces of code within certified devices

order to enable flexible services, it is necessary to let users upload pieces of code (which will be called *applets* in this paper) to servers. Using trust information when deploying the application implies new constraints when distributing data and code. This paper focuses on the implications of this environment for satisfying to *data integrity*, *data confidentiality*, *integrity of execution* and *confidentiality of execution*. In this model, integrity of execution means that servers do not alter the execution of the application and surreptitiously modify its results. Confidentiality of execution aims at preventing the disclosure of program semantics.

2.2 Enforcing Distributed Access Control with Nomadic Applications

The very nature of nomadic computing, in which interactions depend on the user location and environment, generally excludes any type of organization, hence of *a priori* trust assumptions about the servers accessed. This paper specifically addresses the *business to employee* (B2E) and *business to business* (B2B) contexts. This makes it simpler to construct a workable trust model. First of all, public key based authentication is possible and meaningful since employees are directly managed by their corporation. In contrast, in an open trust model, there will generally be no authentication (or trust) infrastructure shared by all entities. Secondly, trust may be based on the partnership established between companies. The trust expectation regarding every partner's tasks and behaviors are contractual and can be translated into a security policy. Trust may also be based on the certification of devices, and specifically their level of tamper-resistance. Again compared with the open trust model, this assumption enables the automation of secure data distribution to different devices.

The following seems realistic to assume in B2E and B2B contexts:

- Each corporation has a local public key infrastructure (PKI).
- Each employee has his/her own asymmetric key pair. Ideally, the private key is protected by a tamper-resistant token (e.g. smart card, SIM card).
- Each device that can be part of a federation has its own asymmetric key pair. The private key can also be protected by a tamper-resistant module.
- Attribute certificates can be delivered to employees so that they can prove their role or that they received some authorizations to other entities.
- Attribute certificates can be delivered to devices so that a device can prove who it belongs to its owner or whether it is tamper-resistant.
- As an extension of signed applets, a set of authorizations can be associated to each piece of code.

In order to protect corporate resources, it is necessary to define access control policies. For instance, the rights of an employee to access corporate data or to use services offered by the environment are easy to implement with public key infrastructures like X.509v3 or SPKI. Relationship agreements between corporations and rights of devices may also be specified similarly.

A contribution of this paper is to use those relationships to define the rights of pieces of code and to distribute data and mobile codes according to the trustworthiness of surrounding devices. Sections 1.1 and 1.2 respectively present existing mechanisms to protect execution environments and pieces of code. The limitations of those approaches are discussed and a new solution is proposed.

3 Specifying Trust Relationships

As explained in the previous section, B2E and B2B applications make it possible to rely on existing trust relationships between companies, employees, and devices to specify a protection adapted to an application. This section shows how those relationships can be defined and verified.

3.1 Attribute Certificates

Attribute certificates have been chosen to formally define relationships and authorizations between the involved actors. The proposed framework uses a proprietary format of attribute certificates. It is an extension of the simple public key infrastructure (SPKI) [EFL+99] that allows to securely associate a capability with an employee or with a device. A public key is embedded in each certificate, and only the user or the device that is in possession of the corresponding private key can use the capability. Rights can be delegated if the certificate allows so and delegation can be performed in a local way without the need to connect to a centralized authority: each user behaves as a local authority for attribute certificates. Delegated credentials have a short lifetime, thus rendering the use of centralized revocation lists unnecessary, and permitting a local validation of the certificate chain. For long-lasting capabilities, revocation lists are envisaged. Attribute certificates are used to store a different type of information: for an employee, it can consist of his role or personal rights; for a device, information about its security-level and the company it belongs to may be provided.

Notations $X \xrightarrow{A,\triangleright,\odot} Y$ means that the entity X certifies some attributes $A = a_1, \cdots, a_n$ of the entity Y. For instance, attributes can define sets of authorizations, security-levels, or roles. In case of authorizations, $\triangleright$ specifies whether those authorizations can be delegated. The clock symbol ($\odot$) refers to short term certificates. A chain of certificates is defined as following:

$$X \overset{A_1,\odot}{\Longrightarrow} Z \quad : \quad X \xrightarrow{A,\triangleright} Y \xrightarrow{A_1,\odot} Z \quad \text{where } A_1 \subseteq A$$

It means that X gives some authorizations $A = \{a_1, a_2, a_3\}$ to Y and allows delegation. Y delegates part of those rights $A_1 \subseteq A$ to Z for a short time, e.g. $A_1 = \{a_1, a_3\}$. It is implemented as a set of SPKI-like attribute certificates (cryptography notations used in this paper are defined in Table 1):

$$X \xrightarrow{A,\triangleright} Y \ : \ \mathrm{SIGN_X}(PK_Y, \mathrm{r} = \{a_1, a_2, a_3\}, \mathrm{d{=}true}, \cdots)$$

$$Y \xrightarrow{A_1,\odot} Z \ : \ \mathrm{SIGN_Y}(PK_Z, \mathrm{r} = \{a_1, a_3\}, \mathrm{d{=}false}, \cdots)$$

PK_A	public key of entity A
SK_A	private key of entity A
$\mathrm{E_K}(m)$	plaintext m encrypted with key K
$\mathrm{h}(m)$	digest of m using hash function h
$\mathrm{SIGN_A}(m)$	plaintext m signed by A i.e. $\mathrm{SIGN_A}(m) = \{m, E_{SK_A}(\mathrm{h}(m))\}$.

Table 1. Cryptographic notations used in this paper.

Section 3.2 defines more precisely the structure of two types of certificate chains: security-level certificates and authorization certificates.

3.2 Authorization and Trust Mechanisms

B2E or B2B context simplifies the trust model. The goal of this architecture is to have a decentralized access control mechanism able to take into account users, companies, and devices. For instance, when an employee accesses a corporate resource from a partner terminal, it is necessary to verify whether this user is authorized to access this resource and whether this device is trusted enough to deal with this resource. Thus it is possible to ensure that a potentially malicious terminal cannot be used to access confidential data. When a piece of code runs on a terminal, it is necessary to verify that the code cannot attack this device.

Authorization capabilities Authorization certificates ensure that a given entity can access a resource when the security-level is sufficient. The set of attributes defines authorizations, i.e. rights (R). The following authorization certificates have been defined:

- $C_i \xrightarrow{R,\triangleright e} C_j$: Company C_i authorizes company C_j to access some resources $R = \{r_1, \cdots, r_n\}$. Those rights can be delegated to employees ($\triangleright e$). For instance, an agreement between C_i and C_j defines that employees of C_j can use some applications of C_i.
- $C_i \xrightarrow{R,<\triangleright e,d>} E_{C_j,m}$: Company C_i authorizes employee $E_{C_j,m}$, who works for company C_j, to access some resources R. Those rights can be delegated to other employees ($\triangleright e$) and/or to devices ($\triangleright d$). When $i = j$ the company delegates rights to its own employee. Example: an employee $E_{C_j,m}$ arrives in company C_i and receives some rights to use local facilities.
- $C_i \xrightarrow{R} P_{C_i,m}$: Company C_i certifies a piece of code and authorizes it to access some resources R. In this case, there is no public key in the certificate but a digest of the code.
- $E_{C_i,m} \xrightarrow{R,<\odot>} E_{C_j,n}$: Employee $E_{C_i,m}$ delegates some rights to employee $E_{C_j,n}$. For instance, a secretary welcomes a visitor and provides him some rights to use local facilities.
- $E_{C_i,m} \xrightarrow{R,\odot} D_{C_j,n}$: Employee $E_{C_i,m}$ delegates some rights to device $D_{C_j,n}$. For instance, an employee authorizes a public terminal to access a corporate document in order to display it.
- $E_{C_i,m} \xrightarrow{R,\odot} P_{C_i,m}$: Employee $E_{C_i,m}$ certifies a piece of code and authorize it to access some local resources R.

A valid chain of authorization certificates is defined as follows:

$$X_1 \overset{R}{\Longrightarrow} X_n \ : \ \{X_i \xrightarrow{R_i,\triangleright t_i} X_{i+1} \quad | \quad 1 < i < n-1\}$$

$$\text{where } R_{i+1} \subseteq R_i \quad 1 < i < n-1 \ \text{ and } \ R = R_{n-1}$$

$$\text{where } t_i \in \{e, d\} \text{ corresponds to } X_{i+2} \quad 1 < i < n-2$$

$C_1 \xrightarrow{R_1,\triangleright e} E_1 \xrightarrow{R_2} E_2$ where $R_2 \subseteq R_1$ is a valid chain. It can mean that company C_1 authorizes its secretary E_1 to delegate some rights, be it using printers or accessing some office, to any visitor. Visitor E_2 received a subset of those rights (e.g. accessing a given meeting room).

Security-level capabilities Security-level certificates are necessary to define the security-level of involved devices that will host code and data. The following security-level certificates have been specified:

- $C_i \xrightarrow{\text{SL}} C_j$: Company C_i trusts devices of company C_j to deal with resources whose classification corresponds to security-level SL. For instance, all devices of a partner company can deal with confidential data related to a given project. It does not mean that devices are authorized to access the resource but that an authorized user can use those devices to access the resource.
- $C_i \xrightarrow{\text{SL}} D_{C_i,m}$: Company C_i authorizes his device $D_{C_i,m}$ to deal with resources whose classification corresponds to security-level SL. For instance, a terminal physically protected can be certified to deal with confidential data.

A valid chain of security-level certificates is defined as follows:

$$C_i \overset{\text{SL}}{\Longrightarrow} D \ : \ C_i \xrightarrow{\text{SL}_{C_j}} C_j \xrightarrow{\text{SL}_D} D$$
$$\text{where SL} = \text{SL}_{C_j} \cap \text{SL}_D$$

$C_1 \xrightarrow{SL_1} C_2 \xrightarrow{SL_2} D_2$ is a valid chain. It can mean that company C_1 trusts a partner company C_2. C_1 accepts that its employees use devices of C_2 to access some type of resource SL_1 e.g. confidential and unclassified data. Company C_2 certifies that a terminal is physically protected and able to deal with some type of resource SL_2.

4 Trust for Resource Protection

Protecting the integrity and confidentiality of resources from potentially malicious code has been widely addressed. However, in nomadic computing systems, new devices can be discovered at any time. It thus becomes necessary to be able to delegate rights when mobile codes, which are parts of an application, are distributed. This section presents an extension of the Java security model that allows defining authorizations in a dynamic and distributed way.

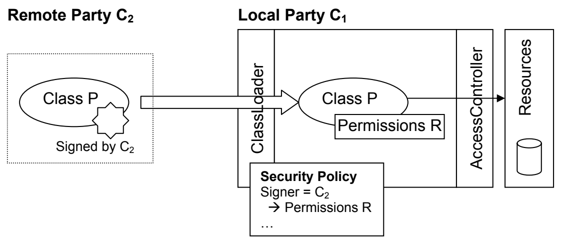

Fig. 2. Signed pieces of code: Java 2 mechanism to protect resources.

The Java 2 security model (as illustrated in Figure 2) is identity based and thus only provides mechanisms similar to access control list to protect resources. [MR00] suggests that instead of signing pieces of code and associating permissions with signers, manipulating capabilities such as chain of authorization certificates associated with pieces of code is required to handle multiple domains in a manageable manner.

4.1 Multiple Domain Resource Protection

Authorization certificates are defined as follows: when company C_1 allows company C_2 to display data on its terminals and C_2 delegates this right to its employee E_2. When E_2 is visiting C_1 he can use the display with his authorization chain:

$$C_1 \overset{display,\triangleright e}{\longrightarrow} C_2 \overset{display}{\longrightarrow} E_2$$

When the employee E_2 has to upload code P_2 to display some data, he provides:

$$C_1 \overset{display,\triangleright e}{\longrightarrow} C_2 \overset{display}{\longrightarrow} E_2 \overset{display,\odot}{\longrightarrow} P_2$$

This chain is verified and converted into Java permissions that are associated to the piece of code. The enforcement of the authorization is thus done by the security manager. Finding a common language to exchange authorizations between companies complicates the setup of this mechanism. At this time, only Java permissions can be encapsulated into certificates. However, Java allows programmers to define their own permissions and thus it is possible to define a common framework that extends this scheme with new permissions such as accessing smart cards or displaying data.

4.2 Reflection-Based Access Control Mechanisms

When a piece of code is loaded, its associated certificate chain is validated. A meta-object protocol (MOP) [KdRB91] is used to intercept all method calls done by this piece of code. The load time MOP "byte code engineering library" (BCEL) [Dah01] has been chosen. Each method call is redirected to a "proxy" that is associated to a protection domain created according to the authorization defined by the certificate chain. Like this, it is easy to dynamically modify the authorizations when a new certificate chain is available. Figure 3 shows how the renewal mechanism works: when a piece of code is uploaded, it comes with some short-term rights defined by the certificate chain. Before the validity end of the certificate $E_2 \overset{R_2,\odot}{\longrightarrow} P_2$, a request for a new chain is sent and the rights of the mobile code are adapted according to the new chain.

Reflection-based access control might allow more complex interactions where, for instance, a piece of code run by a terminal can access local resources (e.g. communication, intranet access, temporary files) as long as the user (or his cellphone) is in front of the terminal.

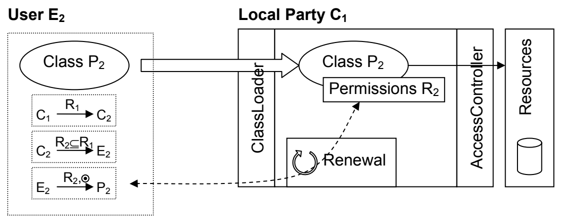

Fig. 3. Certified pieces of code with renewal mechanism.

5 Trust for Execution Protection

In this section, we propose to distribute data and code according to the security-level of federated devices. Securing federations thus becomes evaluating the security-level of each platform that takes part in the federation.

5.1 Security-Levels in B2E/B2B Applications

This evaluation is not easy to achieve in general: if a person makes use of a terminal in a public place, it is impossible to assume that the terminal is trusted in any way without some additional information that makes up the trust model. In general, there is no relation that can be exploited between the user or his company and the owner of the terminal: this can be called an *open trust model* as opposed to an *a priori trust model*. B2E and B2B assumptions provide a clear trust model and allow validating whether a given device is trustworthy (e.g. managed by a partner company, patches are regularly applied). This information is used to distribute code and data according to the security-level of each federated device.

The security-level of a given device depends on the service provider (in B2E or B2B scenarios, a corporate backend server). For instance, a device can be trusted to deal with secret data of the company it belongs to while the same device can only deal with unclassified data when accessing corporate data of another company.

For the sake of simplicity, only three security-levels are used in this section $SL = \{\text{uncl}, \text{conf}, \text{secr}\}$:

- *unclassified (uncl)*: a device with security-level 'uncl' can run pieces of code tagged as unclassified. It can also deal with unclassified data.
- *confidential (conf)*: access to code and data tagged as confidential.
- *secret (secr)*: access to code and data tagged as secret.

Security-levels are defined by a chain of authorization certificates. It is possible to increase the granularity of security-levels by defining new semantics taking into account project names, groups, etc.

5.2 Security-Level Verification

We use security-level capabilities defined in Section 3.2 to provide the functionality described above. For instance, device D is owned and managed by company C (say for instance a wall-display in a meeting room that is physically protected). Company C provides with device D a capability $C \xrightarrow{SL_D} D$.

Agreements between companies are necessary to formally define trust relationships. Such agreements are finally implemented through the issuance of certificates. Company C_2 is a partner of C_1. Employees of C_2 frequently need to work in C_1's offices and use local facilities D_1. Because C_2 trusts C_1, they can have the following trust relationship: $C_2 \xrightarrow{SL_{C_1}} C_1$. When $SL_{C_1} = \{\text{uncl}, \text{conf}\}$, it means that employees of C_2 can use devices owned by C_1 to deal with confidential and unclassified data.

The security-level of each federated device is evaluated thanks to the chain:

$$C_2 \xrightarrow{SL_{C_1}} C_1 \xrightarrow{SL_{D_1}} D_1$$

The security-level of D_1 is defined as $SL_{C_1} \cap SL_{D_1}$. It is important to note that there is no delegation between certificates of this chain and thus $SL_{D_1} \not\subseteq SL_{C_1}$. Moreover, the length of this chain is restricted because agreements are signed partner by partner and cannot be delegated, that is, such agreements do not form a web of trust.

In the previous example, only the owner of a device is involved in the certification process. It is however possible to define more precise trust relationships involving other parameters such as tamper-resistance, location, etc. This mechanism makes it possible to take into account the availability of trusted platform in corporate security policy. For instance, suppose that the owner certifies the device $C \xrightarrow{\text{conf}} D$ and the manufacturer M certifies its tamper-resistance $M \xrightarrow{\text{TR}} D$ where TR is the tamper-resistance level. The security policy could define that confidential data can be read on devices owned by partners with a given security-level or by any neutral tamper-resistant device (e.g. a TCG public terminal).

5.3 Fine Grained Application Deployment

On one hand, employees would like to transparently use any surrounding appliance such as a larger display embedded in a plane seat, a printer in an airport lounge, or location services offered by a building. On the other hand, the corporation has to protect its resources and prevent that an employee unintentionally reveals corporate data to untrusted and potentially malicious devices. Our approach makes it possible a tradeoff between flexibility and security. The deployment of pieces of code is based on the corporate security policy that defines whether a device certified by a given entity can be involved when getting access to non-public data or can run some part of an application. Such an open federation, in which devices owned by different entities are used, will have to be restricted to secure interactions, and yet remain fully usable.

Figure 4 presents a simple example: User E_1 is an employee of C_1 and extends his cell-phone (D_1) with a wall display (D_2). The wall-display is owned and managed by another company C_2. The cell-phone is certified by C_1 and can deal with confidential resources. The wall-display is certified by C_2 and can also deal with confidential data. However, the trust relationship between both companies define that only unclassified data of C_1 can be displayed on devices managed by C_2. As a result, D_1 can deal with confidential resources but D_2 can only deal with unclassified resources.

In order to process some data (e.g. corporate workflow), it is necessary to download data and code (P_1 and P_2). The user authorizes the wall display to get the pieces of code. Both pieces of code are sent to the wall display in an encrypted form. P_1 is tagged as unclassified and thus can be decrypted by D_1 and D_2. P_2 is confidential and can only be decrypted by D_1. In other words, part of the data and part of the code cannot be used by D_2 because it is not trusted enough. For instance, the wall display cannot generate session keys, cannot ask for an employee's signature, cannot display confidential data, etc. Encryption on the corporate server side ensures the enforcement of mandatory access control to corporate resources (access control in Figure 4).

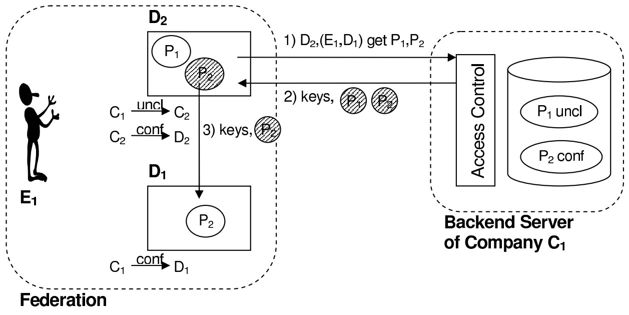

Fig. 4. Distribution of code, data, and keys

5.4 Key distribution

Key distribution ensures that only devices with a sufficient security-level can access (i.e. decrypt) some data and pieces of code that have a given classification. Suppose that a set of pieces of code $P = \{P_1, P_2, \cdots, P_n\}$ are requested from C_1's server by an employee E_1 using a federation $F = \{D_1, D_2, \cdots, D_m\}$.

Data and code are tagged with a different classification $\mathrm{cl}(P_i) \in \mathrm{CL}$, $1 < i < n$. For the sake of simplicity a direct mapping between security-levels and classification has been chosen in this example. For instance, $CL = SL = \{\mathrm{uncl}, \mathrm{conf}, \mathrm{secr}\}$. A symmetric key K_{cl_i} has to be defined for each classification $cl \in CL$:

$$\forall cl \in CL \quad : \text{Server generates symmetric key } K_{cl}$$

Code and data are encrypted according to their classification using the corresponding symmetric key K_{cl}:

$$\forall p \in P : \text{Server encrypts piece of code } \widehat{p} = E_{K_{cl(p)}}(p)$$

The backend server has received the credential of the user and of each federated device. The security-level of each federated device D_i is defined as $\mathrm{tl}(D_i)$. A chain of certificates has to be resolved for each device in order to verify their security-level and distribute keys. Symmetric keys are encrypted with public keys of devices:

$\forall d \in F \quad \text{and} \quad \forall cl \in CL \quad :$

$\quad$ if $cl \in tl(d)$: Server computes $E_{PK_d}(K_{cl(d)})$

The result is that each federated device can potentially receive any encrypted data or any encrypted piece of code. However, it only receives keys for decrypting the parts it is authorized to deal with (Figure 4). For instance, a terminal that is trusted enough to deal with confidential data will receive $K_{confidential}$ and $K_{unclassified}$ but will not receive K_{secret}. Data confidentiality, data integrity, integrity of execution, and confidentiality of execution are enforced by key distribution. Code, data and key distribution is done simultaneously in an XML document.

Discussion and Conclusion

Protecting the integrity of execution of a program in a distributed and potentially malicious environment has long been recognized as difficult, which might appear as a major issue now that nomadic systems promote the use of these very techniques for implementing applications. Fortunately enough, the assumptions of B2E and B2B applications help solve many of the issues encountered for they introduce trust relationships that make it possible to implement a secure system in a pragmatic way.

The core of the framework described in this paper has been implemented. Attribute certificates, which define authorizations and security-levels, are signed XML documents that can be reified as Java objects. Personal Java that we use on Pocket PC (iPAQ) with Bluetooth offers the same security features as Java 2.

The environment protection mechanism (Section 4), which relies on Java 2, associates rights with a set of Java classes and makes it possible to change those rights dynamically based on a new chain of authorization certificates. This approach however imposes that rights be associated with classes, and it might be fruitful to allow object specific rights. A class loader defining distributed and dynamic rights based on short-term certificates has to be finalized. Extending this work in order to associate rights to objects would however mean modifying the virtual machine and thus loosing portability.

The protection of data and code (Section 5) is based on the relationships between device owners. In the current implementation, each device keeps its private key in a key store. Employees' private keys are protected by a trusted device, for instance the employee's PDA. To enforce the mandatory access control, we use a SIM card that becomes a ubiquitous security token in nomadic scenarios. The protection of the user's keys (private key and distribution secret keys) by a SIM card has been implemented but is not integrated. The data and key distribution mechanism has been successfully tested within a prototype that aimed at selectively accessing corporate e-mails from federated terminals according to their security-level. The security-level evaluation of devices has been demonstrated [BRKC03].

This framework also enables the distribution of parts of an application under the form of mobile code by providing the support to writing explicitly independent modules with different security requirements. We are currently studying how a meta object protocol [KdRB91] could be used to provide language support to automatically split and distribute parts of an application according to policy-based security requirements.

The distribution of certificates is a difficult issue: currently certificates are pre-fetched in devices that can create new certificates by delegation. There is a tradeoff between the computational cost of generating certificates on a PDA (i.e. XML parsing, signature) and the communication cost of requesting certificates from a server. We are studying web services (WS-Federations) in order to define protocols for distributing and managing capabilities.

The integration of those different security components is under way and our current focus is on higher level policies to make the creation and management of certificates more flexible. For instance, some users could be authorized to define whether a partner is trustworthy in terms of code creation and/or code hosting. Or it should be possible to take hardware certification (e.g. Trusted Computing Group) into account when evaluating the security-level of a device.

References

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998.

[BFK99] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. The role of trust management in distributed systems security. In *Secure Internet Programming*, pages 185–210, 1999.

[BRKC03] L. Bussard, Y. Roudier, R. Kilian Kehr, and S. Crosta. Trust and authorization in pervasive b2e scenarios. In *Proceedings of the 6th Information Security Conference (ISC'03)*, LNCS. Springer, 2003.

[BV99] Ciaran Bryce and Jan Vitek. The JavaSeal mobile agent kernel. In *First International Symposium on Agent Systems and Applications (ASA'99)/Third International Symposium on Mobile Agents (MA'99)*, Palm Springs, CA, USA, 1999.

[CTL96] C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscating transformations. Technical Report Technical Report 148, Department of Computer Science, University of Auckland, 1996.

[Dah01] M. Dahm. Byte code engineering with the bcel api. Technical Report B-17-98, Freie Universität Berlin, Institut für Informatik, 2001.

[EFL$^+$99] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. Rfc 2693 – spki certificate theory, 1999.

[ENT$^+$02] C. English, P. Nixon, S. Terzis, A. McGettrick, and H. Lowe. Dynamic trust models for ubiquitous computing environments. In *Workshop on Security in Ubiquitous Computing at UBICOMP'2002*, 2002.

[GMPS97] L. Gong, M. Mueller, H. Prafullchandra, and R. Schemers. Going beyond the sandbox: An overview of the new security architecture in the Java Development Kit 1.2. In *USENIX Symposium on Internet Technologies and Systems*, pages 103–112, Monterey, CA, 1997.

[KdRB91] Kiczales, des Rivieres, and Bobrow. *The Art of the Metaobject Protocol.* MIT Press, 1991.

[KFP01] L. Kagal, T. Finin, and Y. Peng. A framework for distributed trust management. In *Workshop on Autonomy, Delegation and Control*, 2001.

[LBR02] S. Loureiro, L. Bussard, and Y. Roudier. Extending tamper-proof hardware security to untrusted execution environments. In *Proceedings of the Fifth Smart Card Research and Advanced Application Conference (CARDIS'02) - USENIX - IFIP working group 8.8 (smart cards)*, 2002.

[LMR00] S. Loureiro, R. Molva, and Y. Roudier. Mobile code security. In *ISYPAR 2000, (4ème Ecole d' Informatique des Systèmes Parallèles et Répartis)*, 2000.

[MR00] R. Molva and Y. Roudier. A distributed access control model for Java. In *6th European Symposium on Research in Computer Security (ESORICS)*, number 1895, pages 291–308, 2000.

[NL98] George C. Necula and Peter Lee. Safe, untrusted agents using proof-carrying code. *Lecture Notes in Computer Science*, 1419, 1998.

[SA02] N. Shankar and W. Arbaugh. On trust for ubiquitous computing. In *Workshop on Security in Ubiquitous Computing at UBICOMP'2002*, 2002.

[ST98] Tomas Sander and Christian F. Tschudin. On software protection via function hiding. *Lecture Notes in Computer Science*, 1525:111–123, 1998.

[SYY99] T. Sander, A. Young, and M. Yung. Non-interactive cryptocomputing for nc1. In *40th Annual Symposium on Foundations of Computer Science 99*, pages 554–566, 1999.

[TCG04] 2004. *Trusted Computing Group (TCG)*, https://www.trustedcomputinggroup.org/home.

[WB97] H. Wasserman and M. Blum. Software reliability via run-time result-checking. *Journal of the ACM*, 44(6):826–849, 1997.

[wit04] 2004. *Wireless Trust for Mobile Business (WiTness)*, IST-2001-32275, http://www.wireless-trust.org.

[Yee99] Bennet S. Yee. A sanctuary for mobile agents. In *Secure Internet Programming*, pages 261–273, 1999.