Protecting Applications and Devices in Nomadic Business Environments

Laurent Bussard and Yves Roudier[†]

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

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 security, execution environment protection, attribute certificates, pervasive computing

Introduction

Computer users are increasingly mobile thanks to the deployment of wireless technologies and to the wide availability of mobile personal devices. Nomadic computing makes it possible for a user 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 worlwide 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 or screen size). For instance, a user travelling 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 which context security becomes a major concern. Indeed, access to the corporate resources and data must be controlled. Furthermore, the safety of the operations performed by a user ultimately depends 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

[†]This research was supported by European Union Project '*WiTness – Wireless Trust for Mobile Business*' [pro04] and by the Institut Eurécom.

code in order to attack the environment hosting it. Otherwise, such appliances might be good candidates for being Trojan horses of a new kind, unbeknownst to their owner.

Application and device 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 addressed 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 objectives 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 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 describes different mechanisms to tackle both problems. More details can be found in research report [BR04].

Protection of execution environments has been widely addressed [LMR00]: *VM approaches* such as the sandbox model, the security model of Java 2 [GMPS97], and JavaSeal [BV99] protect the environment through the isolation of potentially malicious code; *Proof-carrying code* [NL98] relies on a proof that the code respects some security policy. Two security requirements specific to nomadic systems are not fulfilled by those approaches: defining rights of a piece of code in a distributed way should make it possible to delegate rights between entities in charge of certifying pieces of code; a mechanism to dynamically change the rights of an application is also necessary. Section 4 proposes an extension of the Java 2 security model addressing these needs.

Protecting a piece of code against the environment that executes it is notoriously difficult, especially without dedicated hardware. Code can be protected by a *trusted hardware*, be it neutral like in the Trusted Computing Group (TCG) [TCG04], or owned, like when mobile network operators provide SIM cards to their customers. [ST98] suggests the use of encrypted functions, yet this is not enough for encrypting an application. [Yee99] suggests the use of proof based techniques. Obfuscation [CTL96] aims at transforming an application into a functionally identical, but much more complex program, yet its security cannot be evaluated in usual terms.

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 the protection of both code and environment. The approach proposed deals with trust defined as authorizations and/or roles of application developers and users and security level of runtime environments. Some works similarly deal with trust by devising distributed policies for managing its definition [KFP01, BFK99]; however, they do not take into account the security-level of execution environments.

2 Nomadic Business Systems: A Security Architecture

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

2.1 Nomadic System Organization

Figure 1 shows how code distribution is done: different parts 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 performed in a trusted enough environment. Each piece of code receives a short-term authorization to access resources (see Section 4), like say, a word processor can call the signature function but a game cannot.

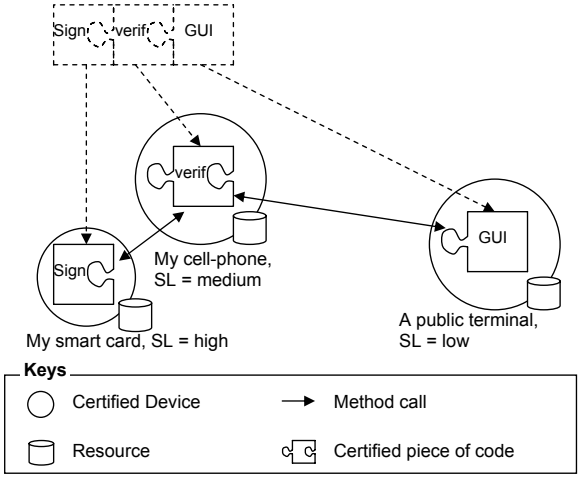

Fig. 1: General overview: certified pieces of code within certified devices

Servers such as public terminals, which are not managed by the user and whose trustworthiness may be questioned, may anyway have to deal with confidential data. Moreover, in 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 vary with the user location and environment, generally excludes any type of organization, hence of *prior* trust assumptions about the servers accessed. This paper specifically addresses the *business-to-employee* (B2E) and *business-to-business* (B2B) contexts, which 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 expectations 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, more specifically their level of tamper-resistance. Again compared with the open trust model, this assumption enables the automation of secure data and code distribution to different devices.

Basic hypothesis in B2E and B2B contexts can be stated as: each corporation has a local public key infrastructure (PKI); each employee has his/her own asymmetric key pair 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, attribute certificates can be delivered to employees so that they can prove to other entities their role or that they received some authorizations; attribute certificates can be delivered to devices so that a device

can prove who it belongs to 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 similarly specified.

The approach described in this paper makes use of those relationships to define the rights of pieces of code and to distribute data and mobile codes according to the trustworthiness of surrounding devices.

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 in an extensible way. 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 locally without the need to connect to a central authority, each user behaving as a local authority for attribute certificates. Using a short lifetime for delegated credentials makes it possible to render the use of centralized revocation lists unnecessary, yet permit a local validation of the certificate chain. For long-lasting capabilities however, revocation lists should be envisaged. Attribute certificates are used to store different types 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 follows:

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

It means that X gives authorizations $A = \{a_1, a_2, a_3\}$ to Y and allows their delegation. Y delegates part of those rights $A_1 \subseteq A$ to Z for a short time, e.g. $A_1 = \{a_1, a_3\}$. This is implemented as a set of SPKI-like attribute certificates:

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

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

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

A B2E or B2B context simplifies the trust model and makes it possible to envisage 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. It is thus 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 may be 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.

- $C_i \xrightarrow{R} P_{C_i,m}$: 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 authorizes it to access some local resources R.

A valid chain of authorization certificates is defined as follows:

$$X_1 \xRightarrow{R} X_n \ : \ \{X_i \xrightarrow{R_i,\triangleright t_i} X_{i+1} \quad | \quad 1 < i < n-1\}$$
$$\text{where } R_{i+1} \subseteq R_i \quad 1 < i < n-1 \text{ and } R = R_{n-1}$$
$$\text{where } t_i \in \{e,d\} \text{ corresponds to } X_{i+2} \quad 1 < i < n-2$$

$C_1 \xrightarrow{R_1,\triangleright e} E_1 \xrightarrow{R_2} E_2$ 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 may be 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 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:

$$C_i \overset{\mathrm{SL}}{\Longrightarrow} D \;:\; C_i \overset{\mathrm{SL}_{C_j}}{\longrightarrow} C_j \overset{\mathrm{SL}_D}{\longrightarrow} D \quad \text{where } \mathrm{SL} = \mathrm{SL}_{C_j} \cap \mathrm{SL}_D$$

$C_1 \xrightarrow{SL_1} C_2 \xrightarrow{SL_2} D_2$ 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.

The Java 2 security model is identity-based and thus only provides mechanisms similar to access control lists 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 Resource Protection with Multiple Domains

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 the following authorization chain:

$$C_1 \xrightarrow{display,\triangleright e} C_2 \xrightarrow{display} E_2$$

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

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

This chain is verified and converted into Java permissions that are associated with 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 it is thus 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 must be validated. A meta-object protocol (MOP) [KdRB91] was used to intercept all method calls done by this piece of code and enforce access control for new types of authorizations. The load time MOP ”byte code engineering library” (BCEL) [Dah01] was chosen for this purpose. Each method call is redirected to a ”proxy” object that is associated with a protection domain created according to the authorization defined by the certificate chain. This makes it easy to dynamically modify the authorizations when a new certificate chain is available. The renewal mechanism works as follows: 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, a request for a new chain is sent and the rights of the mobile code are updated 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 cell-phone) is in front of the terminal.

5 Trust for Execution Protection

This sections discusses how 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 participates to a 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. B2E and B2B applications 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, etc.). This information is useful for distributing 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 $SL = \{\text{uncl}, \text{conf}, \text{secr}\}$ where *unclassified (uncl)* defines a device that can only run pieces of code tagged as unclassified and only deal with unclassified data, *confidential (conf)* gives access to code and data tagged as confidential, and *secret (secr)* enables a full access.

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 Verification of the Security Level

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 physically protected meeting room). Company C provides device D with 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: $C_2 \xrightarrow{SL_{C1}} C_1$. 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 $SL_{D_1} \nsubseteq SL_{C_1}$. 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 platforms 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 only 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 in a corporation

headquarters. On the other hand, the corporation has to protect its resources and prevent that an employee unintentionally reveal corporate data to untrusted and potentially malicious devices. Our approach makes it a tradeoff between flexibility and security possible. 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, yet remain fully usable.

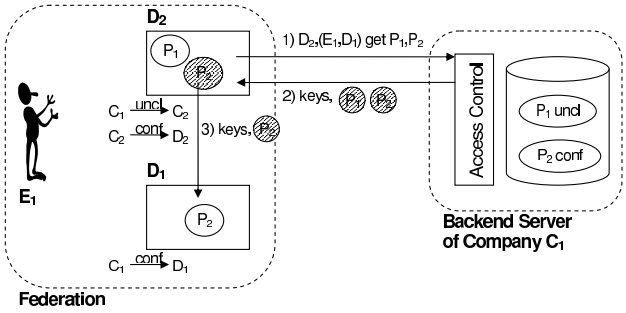

Fig. 2: Distribution of code, data, and keys

Figure 2 illustrates 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 data (e.g. in a corporate workflow), it is necessary to download data and code (P_1 and P_2). The user authorizes the wall display to get the necessary pieces of code. Both pieces of code are sent to the wall display in an encrypted form. P_1 is tagged as unclassified and can thus 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 2).

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 $cl \in CL$:

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

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

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

The backend server has received the credentials 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:

$$\text{for all } d \in F \quad \text{and for all } cl \in CL \quad : \qquad \text{if } cl \in tl(d) : \text{Server computes } E_{PK_d}(K_{cl(d)})$$

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 2). 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 solving many of the issues encountered for they introduce trust relationships that make it possible to implement pragmatic security mechanisms.

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, which 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 requires 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 a mandatory access control, we use a SIM card, which can be seen as 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, as discussed in [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 to 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 and storage 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 render 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. Hardware tamper-resistance certification might also be taken into account when evaluating the security level of a device.

References

[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.

[BR04] L. Bussard and Y. Roudier. Protecting Applications and Devices in Nomadic Business Environments. Technical Report RR-04-101, Institut Eurécom, 2004.

[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.

[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.

[pro04] WiTness project. Wireless Trust for Mobile Business, 2004. IST-2001-32275, http://www.wireless-trust.org.

[ST98] Tomas Sander and Christian F. Tschudin. On Software Protection via Function Hiding. *Lecture Notes in Computer Science*, 1525:111–123, 1998.

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

[Yee99] Bennet S. Yee. A Sanctuary for Mobile Agents. In *Secure Internet Programming*, pages 261–273, 1999.