Cryptographically Protected Objects

Uwe G. Wilhelm wilhelm@lse.epfl.ch
Ecole Polytechnique Fédérale de Lausanne
CH-1015 Lausanne (Switzerland)

1 Introduction

New approaches for distributed computing based on mobile object technology, such as *Java*[1] or *ActiveX*[2] become ever more pervasive and constitute an interesting domain for research that also has important economical implications. However, the usage of the mobile objects paradigm holds certain dangers, such as rogue applications causing damage to the local execution environment, or trojan horses spying out data available in the host system. These dangers are currently being addressed within the mentioned environments (Java security architecture [3], Microsoft authenticode technology [5]).

Other problems, concerned with protecting mobile objects against reverse engineering or contained data from disclosure, are largely ignored. This is partly due to the fact that many people do not consider these problems a real threat[3], but the major issue is that there is no simple solution available. In order to tackle this problem, we need a new approach, since the existing structures can not be adapted to cope with the problem.

In this paper we will present a new approach that guarantees the integrity of the execution environment to the provider of the mobile objects and protects the code and data of the objects against tampering, manipulation, and disclosure, both in transit and during execution.

A rather similar approach for a different environment is described by Herzberg and Pinter in [4]. Their concern is for protecting software sold through regular distribution channels against piracy. The resulting protocols are quite similar, but the technical development has caused rather drastic changes to the computing environment so that a reevaluation of these ideas and an adaption to a more current context seems necessary.

The remainder of the paper is organized as follows. First, in Section 2, we will give a more precise definition of the problem we are concerned with. In Section 3 we introduce some basic definitions and notations that we need for the description of the protocol. Next, in Section 4, we introduce the *cryptographically protected objects* (CryPO) protocol in detail. Finally, Section 5 concludes the paper with a summary of the main contributions and future extensions.

2 The Problem

An *object*[4] O, identified with its unique name O_{name}, consists of code and data that can be sent via a communication channel to a different host. There it can subsequently be executed on a virtual machine, which means that an object is platform independent.

A provider who sends an object to a user currently has no guarantee whatsoever concerning the confidentiality of the code or the data that comprise the object. If the communicated information constitutes a considerable economic value (e.g., proprietary algorithms), there is a potential problem. The receiver of the object could use it in a different context, without the consent or even knowledge of the object provider. The user could also reverse engineer the code to obtain the original object source and reuse all or parts of the illicitly obtained code. Some authors argue that this does not constitute a real problem since the important parts of the code can be copyrighted or should not be distributed in the first place. However, many others consider it a problem since copyright protection can

[1] *Java* is a trademark of Sun Microsystem Ltd.
[2] *ActiveX* is a trademark of Microsoft Corp.
[3] The problem was very controversially discussed on the newsgroup `comp.lang.java.security`.
[4] We use the word object instead of the more verbose *mobile object*. In Java this would be an *applet*; in ActiveX a *control* or a *script*.

be quite tedious (and expensive) and argue that proper protection would be easier to handle. For the latter we propose the *cryptographically protected objects* (CryPO) protocol, which can be the base for a solution to the described problem.

3 Basic Definitions

In order to describe our system, we need to rely on some important concepts from cryptography and tamper resistance.

3.1 Cryptography and Notation

A detailed description of cryptography and the corresponding notation is not within the scope of this presentation. Several books [7, 2] deal extensively with these questions. We will only introduce some basic notations needed for the description of the CryPO protocol.

The protocol uses public key cryptography (such as RSA [6]), in which a *principal* P has a pair of keys (K_P, K_P^{-1}) where K_P is P's public key that is known to everyone and K_P^{-1} the private key that is exclusively known to P. Given these keys and the corresponding algorithm, it is possible to *encrypt* a message m, denoted $\{m\}_{K_P} = m'$, so that only P can *decrypt* m' using his private key: $\{m'\}_{K_P^{-1}} = m$.

Some public key cryptosystems (notably RSA) also allow to use the same keys for *signing* messages. A message m signed by P is denoted $\{m\}_{K_P^{-1}} = m_{sig}$. The signature on m can be verified by everyone using P's public key: $\{m_{sig}\}_{K_P} = m$. If a message, which confers certain rights, bears the signature of a trusted principal it is also referred to as *certificate*.

3.2 Tamper Proofedness

The concept of tamper proofedness usually applies to a well-defined module, sometimes called black-box, that executes a given task. The outside environment cannot interfere with the task of this module other than through a restricted interface, which is under the control of the tamper proof module.

The actual construction of such a tamper proof module in the real world is extremely difficult, nevertheless, there are many applications that rely on them (e.g., payphones, debit cards, or SIM cards for GSM). Given sufficient time and resources, it becomes very probable that an attacker can violate the protection of the module [1]. Therefore, a tamper proof module should not protect information that is more valuable than the estimated cost[5] for breaking its protection. For the remainder of this presentation, we make the explicit assumption that the investment necessary to break our system is much greater than the possible gain.

4 The Overall Approach

In this Section, we will introduce the protocol that guarantees the integrity of the execution environment and protects the code and the data of an object executing in this environment.

We will first present the execution environment that we rely on and then describe the CryPO (cryptographically protected objects) protocol that uses it. Figure 1 gives an overview of the entities in the system and their interactions.

4.1 The Execution Environment

In order to achieve our goal, we need a special execution environment based on the notion of tamper proofedness, which we will call *tamper proof environment* (TPE). The TPE provides a full execution environment for objects, which can not be inspected or tampered with.

The TPE is a complete computer that consists of a CPU, RAM, ROM, and non-volatile storage (e.g. hard-disk or flash RAM). It runs a virtual machine (VM) that serves as execution environment for objects and an operating system that provides the external interface to the TPE and controls the VM (e.g., protection of objects from each other). Furthermore, the TPE contains a private key K_{TPE}^{-1} that is known to no entity other than the TPE – even the physical owner of the TPE has no information concerning this private key (see 4.2).

The TPE is connected to a host computer that is under the control of the TPE owner. This host computer can access the TPE exclusively through a well defined interface that allows, for instance, the following operations on the TPE:

- upload, migrate, or remove objects;

[5] This cost can only be a rough estimation and is likely to decrease over time, when more efficient attacks become possible.

- facilitate interactions between host and object or between different objects on the TPE;
- verify certain properties of the TPE (such as which objects are currently executing).

Due to its implementation as tamper proof module and the restricted access via the operating system, it is impossible to directly access the information that is contained on the TPE.

This property is ensured and guaranteed by the TPE manufacturer[6] *TM*, which also provides the object user OU with a certificate (signed by TM). The certificate contains information about the TPE, such as its manufacturer, its type, the guarantees provided, and its public key. The object provider OP has to trust TM that the TPE actually does provide the protection that is claimed in the certificate.

4.2 Key Generation

One of the most important properties of the TPE that has to be guaranteed with utmost certainty is that the private key of the TPE is never disclosed. To ensure this property, the TPE could create the key itself (based on some random input). Using this approach, the key is never available outside of the TPE and, thus, protected by the operating system and the tamper proofedness of the TPE. This initialization can be done by the TPE manufacturer and subsequently the TPE will be prevented from ever disclosing this key or accepting a new one.

4.3 CryPO Protocol

The CryPO (cryptographically protected objects) protocol transfers objects exclusively in encrypted form over the network to a TPE. Therefore, it is impossible for anyone who does not know the proper key to obtain the code or data of such an encrypted object. By adding a message digest to the encrypted object, the protocol could easily and transparently be extended to also provide integrity protection. Due to space constraints we will not discuss this issue any further.

The protocol is divided into two distinct phases. The first phase consists of an initialization, which has to be executed once before the actual execution of the protocol. The second phase is concerned with the usage of the TPE. The protocol is based on the interactions given in figure 1.

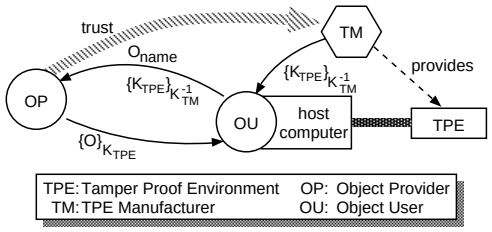

Figure 1: Overview of the CryPO protocol

4.3.1 Initialization

In the initialization phase, the participants establish the trust relations that are associated with the different keys:

- the TM publishes its certification key K_{TM};
- the TM sends the certificate $Cert_{TPE} = \{K_{TPE}\}_{K_{TM}^{-1}}$ to the OU.

4.3.2 TPE Usage

After the participants have finished the initialization, they can execute the actual CryPO protocol:

- The OU sends a request to the OP that contains the name O_{name} of the object he wants to obtain together with the certificate $Cert_{TPE}$ of his TPE.
- The OP verifies the certificate $Cert_{TPE}$ to check the manufacturer and the type of the TPE, in order to decide if it satisfies the security requirements of the OP. Then the OP verifies other information associated with the interaction (such as OU's identity, payment, or others). If the OP is satisfied with these checks, it will continue with the protocol.
- The OP sends the object encrypted under the public key of the TPE, $\{O\}_{K_{TPE}}$ to OU.
- The OU cannot decrypt $\{O\}_{K_{TPE}}$ nor can he do anything other than upload it to his TPE.

[6] It would be possible to extend the model and introduce a certifier that takes some of the responsibilities (e.g. regular controls).

- The TPE decrypts $\{O\}_{K_{TPE}}$ using its private key K_{TPE} and obtains the executable object O. Depending on the directives of OU, O will eventually be started on the TPE and can interact with the local environment of the OU, other objects on the TPE, or even with remote applications.

4.4 What is achieved

The protocol described above guarantees the integrity of the execution environment. This guarantee is based on the trust relation between the OP and the TM. The OP trusts the TM to properly manufacture its TPEs so that the claimed guarantees hold, even without any physical control or access to TPE. Thus, the objects are protected against reverse engineering as well as tampering, manipulation, and disclosure of code or data both in transit and during execution.

The OU is guaranteed that the object can only access the host computer through the specified interface. Any damage that could possibly be caused by an object is limited to the TPE and the restricted access it has to the user's host computer. The TPE could be extended to incorporate an object certification protocol such as those defined for Java or ActiveX [3, 5].

4.5 What can go wrong

The proposed protocol has a weak point that we have to be very careful about. If the secret key of a TPE becomes available to some user, that user can pretend to run a TPE (using the certificate for the TPE of which he has the private key), while he is actually capable to decipher everything that is sent to the TPE. This would violate all the security guarantees offered by the CryPO protocol with a slim chance of that user being discovered.

To alleviate this problem we could require regular verifications of the TPE hardware by some independent certifier. If we can ensure that an attacker cannot obtain the private key from the TPE without destroying it in the process, he would not be able to present the undamaged TPE to the certifier and thus, the duration during which damage could occur would be limited over time.

5 Conclusion

In this paper we introduced a tamper proof execution environment and a protocol that provides a comprehensive protection for the objects executed on this environment. It prevents reverse engineering of the object code and protects the code and data of the object against tampering, manipulation, and disclosure.

The execution environment could be implemented with currently available technology (e.g. using PC-cards such as PCMCIA), although not with the required level of tamper proofedness. Further research in this domain is necessary.

An extension to the CryPO protocol could be used for privacy protection issues. The interface to the TPE could allow objects on the TPE to receive messages. These messages could be encrypted with the public key of the TPE, so that no other entity can access the contents of such a message – not even the owner of the TPE. Furthermore, such a message can be sent to any object executing on the TPE. Assuming that there are several objects on the TPE and none of them indicates that it has received a message, it is impossible to detect which of the objects was the actual receiver of the message. This approach can be used to implement receiver anonymity for the objects on the TPE. We will explore this approach in our future work in the context of telecommunications systems.

References

[1] R. Anderson and M. Kuhn. Tamper resistance — a cautionary note. In *The Second USENIX Workshop on Electronic Commerce Proceedings*, pages 1–11, Oakland, California, November 1996.

[2] G. Brassard. Modern cryptology – a tutorial. volume 325 of *Lecture Notes in Computer Science*. Springer Verlag, 1988.

[3] J. S. Fritzinger and M. Mueller. Java security. White paper, Sun Microsystems, Inc., 1996.

[4] A. Herzberg and S. S. Pinter. Public protection of software. In *Advances in Cryptology: CRYPTO'85*, pages 158–179, Santa Barbara, California, August 1985.

[5] P. Johns. Signing and marking ActiveX controls. *Developer Network News*, November 1996.

[6] RSA Data Security, Inc. *PKCS #1: RSA Encryption Standard.* RSA Data Security, Inc., November 1993.

[7] B. Schneier. *Applied cryptography.* Wiley, New York, 1994.