SMARTCOP – A Smart Card Based Access Control for the Protection of Network Security Components*

Joaquín García-Alfaro[1], Sergio Castillo[1], Jordi Castellà-Roca[2], Guillermo Navarro[1], and Joan Borrell[1]

[1] DEIC/UAB, 08193 Bellaterra (Catalonia), Spain
Email: {jgarcia,scastillo,gnavarro,jborrell}@deic.uab.es
[2] DEiM-ETSE-URV, 43007 Tarragona (Catalonia), Spain
Email: jordi.castella@urv.net

Abstract. The protection of network security components, such as *firewalls* and *Intrusion Detection Systems*, is a serious problem which, if not solved, may lead a remote adversary to compromise the security of other components, and even to obtain the control of the system itself. We are actually working on the development of a kernel based access control method, which intercepts and cancels forbidden system calls potentially launched by a remote attacker. This way, even if the attacker gains administration permissions, she will not achieve her purpose. To solve the administration constraints of our approach, we use a smart card based authentication mechanism for ensuring the administrator's identity. In this paper, we present an enhanced version of our authentication mechanism, based on a public key cryptographic protocol. Through this protocol, our protection module efficiently verifies administrator's actions before granting her the privileges to manipulate a component.

1 Introduction

Network security components, such as and *firewalls* and *Intrusion Detection Systems*, are almost always working with special privileges to execute their tasks. This situation can allow remote attackers to acquire these privileges and perform unauthorized activities [2]. The existence of programming errors within the code of these components, the illicit manipulation of their related resources (e.g., processes, executables, and configuration files), or even the increase of privileges through operating system's errors, are just a few examples regarding means in which a remote adversary can bypass traditional security policy controls.

In [4] we presented a protection module integrated into the kernel of an attack prevention system intended to intercept and cancel forbidden system calls launched by a remote attacker. More specifically, the mechanism we presented avoids escalation attacks through an access control scheme which handles the protection of the system's elements. Indeed, this scheme prevents that potentially dangerous system calls (e.g., cancellation of a process) could be produced from one element against another one. The

* This work has been partially funded by the Spanish Ministry of Science and Technology (MCYT) through the projects TIC2003-02041 and SEG2004-04352-C04-01, and the Catalan Government Department DURSI, with its grant 2003FI-126.

protection is hence achieved by incorporating an access control mechanism that may allow or deny a system call based on several criteria – such as the identifier of the process making the call or some of the parameters passed to it.

The approach presented in [4] allows, moreover, to keep away from the necessity of trusting special users with privileged rights, by delegating the authorization for the execution of a given system call to the internal access control mechanisms. Therefore, and contrary to other approaches, it provides a unified solution, avoiding the implementation of different specific mechanisms for each component, and enforcing the compartmentalization principle [10]. This principle is based in the segmentation of a system, so several elements can be protected independently one from another. This ensures that even if one of the elements is compromised, the rest of them can operate in a trusted way. For our job, several elements from each component are executed as processes. By specifying the proper permission based on the process ID, for instance, we can limit the interaction between these elements of the component. If an attacker takes control of a process associated to a given component (through a buffer overflow, for example), she will be limited to make the system call for this given process.

Nevertheless, it is not always possible to achieve a complete independence between the elements. There is a need to determine which system calls may be considered as a threat when launched against an element from the component. This requires a meticulous study of each one of the system calls provided by the kernel of a given operating system, and how they can be misused. We must also define the access control rules for each one of these system calls. For our approach, we proposed the following protection levels to classify the system calls: (1) critical processes protection; (2) communication mechanisms protection; and (3) protection of files associated to the elements.

According to these protection levels, we then presented in [5] a prototype implementation of our kernel based access control mechanism developed for *GNU/Linux* systems and called SMARTCOP (which stands for *Smart Card Enhanced Linux Security Module for Component Protection*). This implementation was developed over the *Linux Security Modules* (LSM) framework [11]. This framework does not consist of a single specific access control mechanism; instead it provides a generic framework, which can accommodate several approaches. It supplies several hooks (i.e., interception points) across the kernel that can be used to implement different access control strategies. Such hooks are: *Task hooks*, *Program Loading Hooks*, *File systems Hooks* and *Network hooks*. This set of LSM hooks can be used to provide protection at the three different levels proposed above.

Furthermore, the LSM framework adds a set of benefits to our implementation. First of all, it introduces a minimum load to the system when comparing it to kernels without LSM, and does not interfere with the normal system activities [11]; second, the access control mechanism can be integrated in the system as a module, without having to recompile the kernel; third, it provides a high degree of flexibility and portability to our implementation when compared to other proposals for the Linux kernel, such as [7] and [9], where the implementation may require some kernel modifications; and fourth, the LSM interface provides an abstraction which allows the modules to mediate between the users and the internal objects from the operating system kernel – to this effect, before accessing the internal object, a hook may call functions provided by a

given module and which may decide whether allow or deny the access to the internal object, for example.

Through the use of SMARTCOP as a LSM module, the component's processes are allowed to make operations only permitted to the administrator officer – such as packet filtering and application cancellation. The internal access control mechanisms at the kernel are based in the process identifier (PID) that makes the system call, which will be associated to a specific element. Each function registered by a LSM module, determines which component is making the call from the PID of the associated process. It then, applies the access control constraints taking also into account the parameters of the system call. Thus, for example, a given element can access its own configuration files but not configuration files from other elements.

Our protection strategy introduces, however, some administration constraints, since officers are not longer allowed to throw system calls which may suppose a threat to the protected component. To solve these constraints, we also presented in [5] a smart card based authentication mechanism, based on secret-key cryptography, which acts as a reinforcement of the kernel-based access control. The objective of this complementary mechanism is twofold. First, it holds to the administrator the indispensable privileges to carry out management and configuration activities just when she verifies her identity through a two-factor authentication mechanism. Second, it allows us to avoid those attacks focused on getting the rights of the administrative entity, such as dictionary-based attacks and buffer overflows.

Nevertheless, and although the authentication mechanism proposed in [5] solves the administration constraints of our approach, it presents important drawbacks. For instance, there is a need for the entities to share a secret-key, and this is a serious disadvantage for the administration officer, who may be in charge of managing such keys. The process of changing or updating the shared secret-key of all the entities, for instance, over the complete set of components of a network will be very awkward, making it even unfeasible when using our authentication mechanism on huge corporation networks with multiple resources to protect. For this reason, and in order to make easier the administration tasks of our protection approach, we extend in this paper our previous authentication mechanism by using a new authentication protocol based on public key cryptography. Indeed, our new proposal solves the administration constraints of SMARTCOP by using a hierarchical structure with several domains, where the nodes of each domain can independently be administrated by using X.509 certificates [12]. Through this new authentication mechanism, some of the previous drawbacks, such as the sharing of the protocol's information, should be more efficiently performed by means of certificate revocation, for example.

The remainder of this paper is organized as follows. We first define in Section 2 the structure and elements for our new authentication proposal, and present the cryptographic protocol intended to solve the administration constraints introduced by the protection approach described above. We then continue in Section 3 by presenting some configuration issues of our proposal and showing the results of an evaluation of the overhead introduced by our approach on a given setup. We finally summarize in Section 4 some related works, and close the paper in Section 5 with a list of conclusions and future work.

2 Smart Card Based Authentication Mechanism

In order to verify the administrator's identity of SMARTCOP, we propose a two-factor authentication mechanism based on the cryptographic functions of a smart card. This mechanism is intended for authenticating the administrator to the LSM modules and holds with the following requirements: (1) the actions must be authorized by the use of a smart card; (2) the smart card only authorizes one action whenever the PIN would be correct; and (3) the LSM module only authorizes the action whenever the smart card response would be valid, i.e., the cryptographic operation is correct.

Let us start the description of our authentication mechanism by introducing the necessary structure and elements for our proposal. We first define the necessary architecture for our authentication protocol as a hierarchical structure with several organizational units, where the network is divided, in turn, in hierarchical domains, and where each domain of the network has several components that must be protected. We name such a component as SMARTCOP Node (SCN). Each domain has moreover a SMARTCOP Server (SCS), and each potential administrator holds a SMARTCOP Card (SCC). These component are briefly described next.

SMARTCOP Server (SCS) – Each SCS owns a cryptographic key pair *master key* and the corresponding certificate. This certificate has been issued by the upper SCS in the hierarchy and identifies the lower SCS as a valid SCS. This certificate is encoded as an X.509 Attribute Certificate [12], where the issuer is the upper SCS master key and the subject is the lower SCS master key. The SCS of domain B can issue certificates authorizing a concrete SCC as an administration of the domain B (similar to the certificates between SCSs). The SCS must usually be managed by the network administration officer of the given domain – or organizational unit. That is, the person who has more knowledge about the network domain and its potential administrators, and, at the same time, the one that has the greatest interest in performing a good administration. This is a key point of the extended authentication proposal, which enables the distribution of the administrative management between domains or organizational units.

SMARTCOP Node (SCN) – Each SCN is a component which has the SMARTCOP LSM module. The security parameters of the LSM module are properly initialized when it is installed. The main parameter is the *Source-of-Authority* (SoA), which is represented by a *master-key*. More precisely, the *master-key* of the top SCS. When an administrator requests a protected action on a given SCN, by using Protocol 1, the SCN verifies the certificate from the SCC. Then, if it comes from a certificate path rooted at the SoA's *master-key*, the operation is accepted.

SMARTCOP Card (SCC) – The SCC is owned by potential administrators. In order to be able to perform administrative tasks on a given domain, the SCC must be authorized (i.e., certified) by the SCS of the domain or an upper one in the hierarchy. Each SCC has a key pair, which has to be certified by a *master-key* (i.e., a key from a SCS). Let us recall that the cryptographic engine of such a smart card is capable of performing several cryptographic functions, such as asymmetric key generation, asymmetric cryptographic algorithms execution, and so on.

The SCC has an *operation PIN* and an *administration password*. The operation PIN is at least six digits long and is used to authorize the protected actions. On the other hand, the administration password is used to change the operation PIN and other management tasks. The system administrator has three consecutive chances to enter the operation PIN. In the third entry, if the smart card receives an incorrect operation PIN, it blocks itself. The smart card can only be unblocked with the administration password. Again, there are three chances to enter the correct administration password. Otherwise, after the failing of three consecutive wrong administration passwords, the smart card blocks itself and becomes useless.

2.1 Protocol Description

We give here a detailed description of the cryptographic protocol that leads our smart card based authentication mechanism. It starts in Step 1 when the system administrator requests an action to the LSM module. We assume here that action X must be authorized by using the smart card. The LSM module blocks immediately in Step 2a the communication channel between the smart card reader and the LSM module. In this way, we can assure that the data sent between the module and the smart card can neither be sniffed nor tampered. The module also forces to remove the smart card when is not necessary. In Step 2c, the LSM module waits for the smart card insertion, and in Step 4e the LSM module does not proceed until the smart card has been removed. In Step 3 the operation PIN travels in a secure way from the keyboard because the LSM module has blocked the channel between the keyboard and the module itself. Then, LSM sends a NONCE obtained at random and the PIN in step 4c. The smart card returns the digital signature of the NONCE computed with the smart card's private key. The protocol concludes in Step 4g where the LSM module verifies whether the digital signature has been computed properly and the digital certificate is valid.

Protocol 1

1. *The system administrator opens a new console and she requests an action X;*
2. *LSM receives the request from the console and it does the following steps:*
   (a) *Block the channel and open a connection with the smart card reader;*
   (b) *Print a message asking to insert the smart card into the reader;*
   (c) *While the smart card has not been inserted do;*
      i. *Detect the insertion of the smart card;*
   (d) *Print a message asking for the operation PIN;*
3. *The system administrator types the operation PIN in the keyboard;*
4. *The LSM does the following steps:*
   (a) *Obtain the operation PIN;*
   (b) *Obtain a NONCE value at random;*
   (c) *Execute the Procedure 1 inside the smart card by using the operation PIN and the NONCE, and obtain a response μ;*
   (d) *Print a message to remove the smart card from the smart card reader;*
   (e) *While the smart card has not been removed do;*
      i. *Detect the removing of the smart card;*

(f) *if* μ *is* ERROR *the LSM does not authorize the action* X*;*
(g) *else do:*
   i. *Check if the digital signature has been computed with a public key, which belongs to a certification path rooted at the* master key *(SoA).*
   ii. *Verify the smart card certificate against a valid CRL.*
   iii. *Verify the digital signature* μ *with the public key* P_K *obtained from the smart card certificate,* $P_K(\mu) \stackrel{?}{=} H(NONCE)$*;*
   iv. *if the verification is correct the LSM authorizes the action* X
   v. *if the verification is not correct the LSM does not authorize the action* X*;*

We show next the procedure that is executed within the smart card (cf. Procedure 1). Through such a procedure, the smart card can validate the operation PIN. Whenever the operation PIN is valid, it computes the digital signature of NONCE with the smart card private key.

Procedure 1 $[PIN, NONCE]$

1. *Validate the operation PIN;*
2. *If the operation PIN is correct do:*
   (a) *Compute the digital signature of NONCE with the private key* S_K*,* $\mu = S_K(NONCE)$*;*
   (b) *return* μ*;*
3. *If the operation PIN is no correct return* ERROR*;*

To ensure the proper execution of both Protocol 1 and Procedure 1, we have also considered the protection of the entities and the channels involved in such a process, avoiding attacks such as impersonation and channel data manipulation. First, the LSM module guarantees that the binary file of the console can not be overwritten by anyone (even the security officer), remaining the permissions as read-only. Second, the console's executable is compiled in a static fashion. This allows us to reduce the complexity of the protection's console process, since we do not need to consider extra tasks introduced by the loading of shared libraries and its associated files. It also allows us to centralize and reduce the failure points that could be used by a remote attacker which tries to tamper the console's process. Third, the LSM module also controls that each system call launched by any other process in the system does not interfere the normal execution flow of the console's process, such as keyboard key capture, cancellation, or debugging process system calls.

It is also important to recall that the communication channel can not be manipulated by any opponent, since the LSM mediates between the system calls related with the communication channels and the entities that take part within the protocol. Furthermore, and as pointed out in [5], the LSM module does not need to be directly protected since we can assume the kernel environment as a trusted area – since it is mandatory for the kernel security model of any modern operating system.

3 Configuration and Performance Evaluation

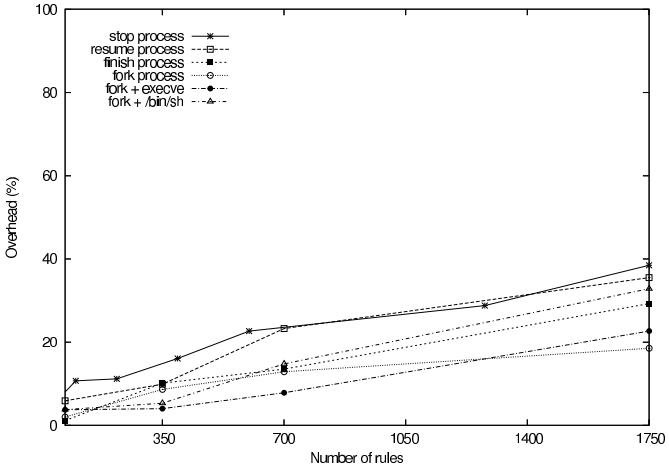
In order to define the objects and resources to protect, SMARTCOP can actually be configured through a set of security rules. Each rule defines an *action* in $\{deny, accept\}$ that applies over a set of *condition* attributes, such as user_id (*UID*), process_id (*PID*), device, i-node, etc. We can also define, through these security rules, either open or closed default policies. The complete set of rules are stored in a set of configuration files that are loaded at boot time through the *proc file system*. The *proc file system* (procfs) is a special virtual file system in the Linux kernel which allows user space programs to access kernel data structures.

Up to now, we have defined different points through procfs for configuring the protection of the three basic levels of protection stated in Section 1. More specifically, we have defined the following entries: *iperms*, *iren*, *isetattr*, *iunlink*, *tcreate*, and *tkill*. The four first labels refer to i-node related operations (resp., i-node permission verification, i-node renaming, i-node permission changing, and i-node removing). They can be used not only for the protection of file resources, but also for the protection of communication operations through, for example, sockets and pipes. The last two labels (i.e., tcreate and tkill) are related to the managing of processes (such as creation, suspending, resuming, termination, and spawning of processes). Through these configuration points, we conducted several tests steered towards measuring the penalty introduced by the installation of SMARTCOP as a LSM module, over the normal operation of the system. The tests and benchmarks were based on LMbench [8] and other related administration tools. The evaluation was carried out on a single machine with an Intel-Pentium M 1.4 GHz, with 512 MB of RAM memory and an IDE hard disc of 5400 rpm, running a Debian GNU/Linux operating system and ext3 file system.

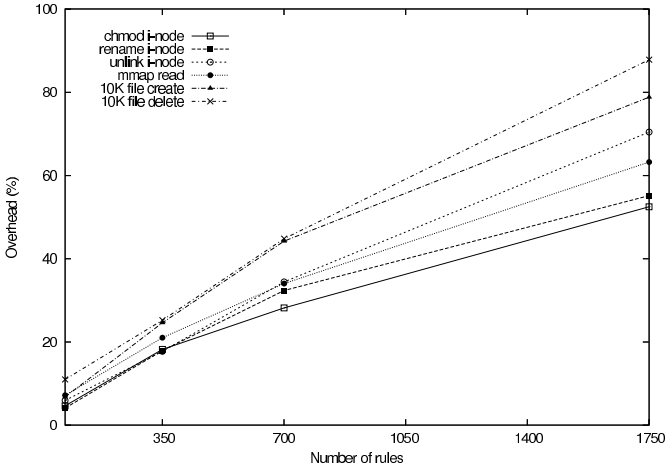
During these tests, we measured the overhead of our approach with an instance of SMARTCOP configured with a closed default policy and loaded with different protection rules. More specifically, each configuration point was charged with a set of auto-generated *accept* rules, initially empty, and which incremented to more than three hundred rules. Therefore, a progressive set of auto-generated *accept* rules from zero to more than one thousand rules was globally loaded. We consider that the overlaps between rules, related with the single operations we measured during these tests, represent the worst case scenario we can actually measure. We show in Figure 1(a) and Figure 1(b) the overhead evolution of some actions that we consider representative regarding the set of configuration points we described above.

The first three curves we show in Figure 1(a) represent the overhead evolution of the system call *kill* when we, resp., suspended, resumed, and cancelled a set of processes under the different load of rules. Notice that such actions, especially when suspending processes, reported an acceptable penalty (aprox. a 40% for a global average of almost two thousand rules). The other three curves in Figure 1(a) represent the overhead evolution of the set of operations related to the creation of processes through *fork()*, *fork()+exec()* and *fork()+/bin/sh*. Notice that the two first operations supposed a penalty even lower (aprox. a 20% for the highest average of rules); and the third operation remained close the 30% for the same number of rules. Similarly, the results we show in Figure 1(b) are related to the evolution of operations for the managing of i-nodes (such as files, pipes, and sockets' managing). We can appreciate in these results, however, that

the penalty introduced by SMARTCOP for the managing of i-nodes seems much higher than the overhead introduced for the managing of processes – it even reached more than an 80% in the operations of file creation and removing. However, we consider that these differences are reasonable, taking into account that there was an overlap between processes protection's rules and i-nodes protection's rules – expressly introduced during our experiments to simulate the worst case scenario. This overlap between rules definitively exercises a bad influence on the measured i-node operations, compared to the processes operations, and it explains the differences between both results.



(a) Processes tests



(b) Filesystem and communication tests

Fig. 1. Performance evaluation of SMARTCOP.

4 Related Work

There are two main approaches to safely execute processes with special privileges on modern operating systems. A first approach is the creation of restricted environments, in which the processes will be executed and controlled outside the trusted system space. In [6], for instance, we can find a traditional mechanism for the creation of restricted environments within Unix setups. These proposals require, however, a replicated file system tree for the protected environments. Hence, the administrator in charge of the system must reproduce the original file system tree to include, for example, shared libraries or configuration files, and copy them to the new path. Other disadvantage of these proposals is that they do not guarantee the correct execution flow of processes, i.e., the behavior of a given process can be modified by using, for example, a buffer overflow. Hence, the attacker can overwrite the configuration or log files of such a process by simply using an arbitrary code execution attack – since these files remain in the same environment of the protected security component process.

A second approach, as the one presented in this paper, is to apply a kernel based access control to outcoming system calls. In [7] and [9], for instance, two similar proposals to ours are presented. The main goal behind these two proposals is to reinforce the complete system by controlling the system calls and ensuring which process or user does the system call and against what it will be done. The ability to control the access to the resources allows to protect system's elements and to avoid that nobody (including an attacker with administrator privileges) can disable them. Nevertheless, both approaches differ from ours in a number of ways. First, and to our best knowledge, neither [7] nor [9] do not address the management of administration constraints, as our proposal does through the two-factor authentication mechanism we present in Section 2. Second, our approach, entirely based on the *Linux Security Modules* (LSM) framework [11], guarantees the compatibility with previous applications and kernel modules without the necessity of modifications. Both [7] and [9] require the rewriting of some features of the original operating system's kernel to properly work. This situation may force to recompile existing code and/or modules in order to obtain the new security features. Although it exists a LSM-based prototype for the approach presented in [7], it does not seem to be actively maintained for the current Linux-2.6 kernel series.

5 Conclusions

We have presented in this paper an access control mechanism specially suited for the protection of network security components, such as *firewalls* and *Intrusion Detection Systems*. Whenever one of these components, or one of its elements, is compromised by a remote attacker, it may lead her to obtain the full control of the network [2]. The protection of these components is not easy, specially when dealing with distributed setups, made up of different elements distributed over a complex network. Like for example, the attack prevention platform presented in [3]. The solution we provided proposes the protection of the components by making use of the *Linux Security Modules* (LSM) framework for the Linux kernel over GNU/Linux systems [11]. The developed mechanism works by providing and enforcing access control rules at system calls, and is based

on a protection module integrated into the operating system's kernel, providing a high degree of modularity and independence between elements. Furthermore, the use of a complementary authentication method, based on smart card technology and a public-key cryptographic protocol, allows us to properly verify administrator's actions when officers need to do administration tasks. This additional enhancement also allows us to prevent some logical attacks against the protection mechanism itself (e.g., password forgery). The integration of our approach on a normal system setup proved, moreover, a good degree of transparency to the administrator in charge, and a reasonable performance penalty for the managing of processes, files, and communication resources.

As a future extension of our work, we are considering improving the customizing of policies. Up to now, the specific policy that is enforced by our protection module is loaded at boot time through the *proc file system* (procfs). We are planning to extend this feature to add the possibility of using text-based configuration files and the reload of policies at runtime. We are also considering to continue our study to address the security of the system from an intrusion tolerance point of view [1].

References

1. Y. Deswarte, L. Blain, and J. C. Fabre. Intrusion tolerance in distributed computing systems. *IEEE Symposium on Security and Privacy*, pages 110–121, Oakland, CA, USA, 1991.
2. D. Geer. Just How Secure Are Security Products? *IEEE Computer*, 37(6):14–16, 2004.
3. J. García-Alfaro, F. Autrel, J. Borrell, S. Castillo, F. Cuppens, and G. Navarro. Decentralized publish/subscribe system to prevent coordinated attacks via alert correlation. *6th Int. Conf. on Information and Communications Security*, 223–235, Spain, 2004.
4. J. García-Alfaro, S. Castillo, G. Navarro, and J. Borrell. ACAPS: An Access Control Mechanism to Protect the Components of an Attack Prevention System. *Journal of Computer Science and Network Security*, 5(11):87-94, 2005.
5. J. García-Alfaro, S. Castillo, J. Castellà-Roca, G. Navarro, and J. Borrell. Protection of Components based on a Smart-card Enhanced Security Module. *1st International Workshop on Critical Information Infrastructures Security, Information Security Conference (ISC'06)*, Samos, Greece, 2006.
6. P. Hope. Using Jails in FreeBSD for Fun and Profit. *Login; The Magazine of Usenix & Sage*, 27(3):48–55, 2002.
7. P. Loscocco and S. Smalley. Integrating Flexible Support for Security Policies into the Linux Operating System. *11th FREENIX Track: 2001 USENIX Annual Technical Conference*, USA, 2001.
8. L. McVoy. LMbench, Portable Tools for Performance Analysis. *1996 USENIX Annual Technical Conference*, USA, 1996.
9. A. Ott. The Role Compatibility Security Model. *7th Nordic Workshop on Secure IT Systems (Nordsec 2002)*, Karlstad University, Sweden, 2002.
10. J. Viega, and G. McGraw. *Building Secure Software - How to Avoid Security Problems the Right Way*. Addison-Wesley, 2002.
11. C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman. Linux Security Modules: General Security Support for the Linux Kernel. *11th USENIX Security Symposium*, USA, 2002.
12. ITU-T. The Directory: Public-key and attribute certificate frameworks. ITU-T Recommendation X.509, 2000.