# Application of Formal Analysis to Enhancing Trust in a Complex Grid-based Operating System

Benjamin Aziz
University of Portsmouth
Portsmouth, United Kingdom
benjamin.aziz@port.ac.uk

## ABSTRACT

This paper presents a case study in the application of formal modelling and verification techniques to a large-scale distributed operating system for Grids called XtreemOS. The process algebraic language of applied $\pi$-calculus is used to model one of the mutual authentication protocols in the XtreemOS trust model, and an associated tool called ProVerif is used to verify the data leakage and mutual authentication properties in the protocol. The results, beside enhancing the level of assurance of the protocol in a critical part of the system, contribute to better understanding of the level of detail in the protocol's specification hence enabling better implementation of the protocol.

## Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*formal methods*; D.4.6 [**Operating Systems**]: Security and Protection—*verification*

## General Terms

Design, Security, Theory, Verification

## Keywords

formal analysis, trust, Grid systems, operating systems

## 1. INTRODUCTION

The management of trust in large-scale distributed systems, such as Grids, is a complex process, which requires various mechanisms and techniques to adapt to the individual components in the system without compromising various security requirements and assurances. Such mechanisms and techniques may include personal vetting, cryptographic credentials and security protocols for the purpose of authenticating communicating entities. Traditionally, the use of formal modelling and verification was adopted to obtain rigorous understanding of the properties that hold in a security protocol [6, 8, 17, 24, 25, 7, 4, 27, 5]. However, by its very nature, formal verification is an expensive process and requires a high level of expertise. Therefore, its application within a complex computing system should preferably be confined to the parts that are considered to be critical.

The work presented in this paper is one example of such use of formal verification techniques in enhancing the assurance about the level of trust in a large-scale Grid-based distributed operating system, called XtreemOS [12, 23]. A process algebraic methodology, called applied $\pi$-calculus [2], and its associated tool, ProVerif [7] are applied to the modelling and verification of a mutual authentication protocol, which is used to establish trusted communications between users and resources in an XtreemOS Grid. As a result, the main significance of this work can be expressed in terms of the following two major contributions:

- A critical part of the XtreemOS trust model as defined by one of its mutual authentication protocols is formally verified in light of desireable security properties for that protocol, therefore increasing its level of assurance within the system.
- As a result of the formal modelling of the protocol, extra detail is added to the original informal specification, which facilitates the implementation stage following the specification.

The formal verification work presented in this paper was part of a larger effort in project XtreemOS for applying formal methods to the improvement of certain parts of the system and at different stages of the development lifecycle. Other examples included [3, 5]. In [5], the application of formal analysis was applied to another protocol in the project for achieving delegation of trust between users and resources. The analysis uncovered various flaws in the original protocol. In [3], formal modelling of some of the security requirements in XtreemOS was used to consolidate their understanding and better direct the design process.

The rest of the paper is organised as follows. In Section 2 we provide a general overview of the architecture of the XtreemOS operating system. In Section 3, we discuss the XtreemOS trust management process and the model of trust it is based on. In Section 4, we introduce one aspect of this trust model, namely a user/resource mutual authentication protocol. In Section 5, we define the formal model for this protocol and apply a formal analysis tool to verify that the model upholds desireable security properties, such as secrecy of data and mutual authentication. Finally, we conclude the paper in Section 7.

## 2. OVERVIEW OF THE XTREEMOS OPERATING SYSTEM

As we mentioned in the Introduction, the main use case motivating our work here was XtreemOS (www.xtreemos.eu) [12, 23], an EU FP6 project, which was aimed at building a Grid-based distributed operating system that provided a single abstraction of physical hardware and software services offered by a collection of standalone Linux operating systems to users within a Grid. These operating systems could function collaboratively to support the utilization of computational and storage resources regardless of the geographical location of their users or machines. A major function of XtreemOS was to hide the complexity of distributed resources dynamically aggregated from large-scale cross-domain resource providers and to ensure the transparency of using such a distributed operating system. Hence, similar to a standalone operating system, once a user is registered with XtreemOS, it should be conceptually the same to utilise resources from any machine that the system is composed of, regardless of whether such resources have been added to the system recently or have been there before.

As illustrated in Figure 1 [12], XtreemOS is composed of two parts: the XtreemOS foundation, called XtreemOS-F, and high-level Grid services, called XtreemOS-G. XtreemOS-F is a modified Linux kernel embedding support for Virtual Organisations (VOs), where a VO in XtreemOS is a temporary collaboration among various Grid resource providers and resources for achieving a specific goal. XtreemOS-F is a modified Linux kernel embedding VO support mechanisms and providing kernel level process checkpoint/restart functionalities. XtreemOS-G comprises several Grid OS distributed services to deal with resource and application management in VOs, and is implemented onto XtreemOS-F.

XtreemOS targets scalable and flexible management of dynamic VOs [11]. XtreemOS Grids spans multiple administrative domains on different sites, comprising heterogeneous resources that can be shared by the participating organisations. A Grid member can create a VO, for which he becomes the VO owner. Any Grid member can request his registration in a given VO, subject to the VO owner approval. Resources can be registered in VOs as well. The VO owner defines policies stating permissions and usage rules for VO resources. Grid administrator also defines policies regulating what a Grid member can do (for example, permission to create a VO). Resources owners in the different administrative domains may also define local policies for resource usage. Grid, VO and local policies are enforced by the XtreemOS system.

The Application Execution Management [9] services are in charge of discovering, selecting and allocating resources for job execution, as well as starting, controlling and monitoring jobs. Data management in XtreemOS is achieved with the XtreemFS Grid file system [28]. XtreemFS federates multiple data stores located in different administrative domains and provides secure access to stored files to VO members, whatever their location.

## 3. TRUST MANAGEMENT IN XTREEMOS

According to Grandison and Sloman [19], *trust* is one aspect of belief in the competence of an entity to act dependably, securely and reliably within a specified context. Trust can be categorised into several classes among which are the *service provision trust* and the *certification trust*. Service provision trust denotes the reliance of a user on the functionality of a service, which is an essential aspect of Grid-based applications. Certification trust, on the other hand, refers to trust built on a set of certified attributes.

Trust in XtreemOS is grounded on three main aspects. First, trust is perceived as an administrative separation of resource and service ownership and management. Different organisations, resource owners, users and core XtreemOS service managers are allocated their own domains, which define their own boundaries of trust. This notion is based on the notion of service provision trust. Second, trust is asserted in special tokens created based on cryptographic mechanisms such as digital certificates and other credentials, which are then verified by their consumers. These tokens convey verifiable attributes of entities that allow them to establish trust with other entities existent in other domains. This aspect is similar to certification trust. Finally, the transmission of trust tokens is achieved via trustworthy communication channels and protocols. This final aspect is the focus of the formal verification work presented in this paper for one of the XtreemOS authentication protocols.

In the following sections, we give a concise overview of the trust model and management process in XtreemOS. More detail can be obtained from [10].

### 3.1 Trust Domains

Domains (or sites) refer to the separation in the ownership and management of software and hardware resources as well as user membership. We use the term *trust domain* to indicate the level of assurance that each domain provides the designers, administrators and users of the XtreemOS system with.

In its broad definition, XtreemOS consists of three main trust domains: *Core sites*, *Resource sites* and *User sites*. Assuming that $S > S'$ is an *assurance ordering relation* taken from some lattice of assurance levels (e.g. [13]) to indicate that $S$ has a higher assurance level (and is therefore more trustworthy) than $S'$, then the ordering among the three trust domains in XtreemOS is as follows: *Core Site* > *Resource Site*, *Core Site* > *User Site*. From this relation, the Core site is required to be more trusted than either the User or the Resource sites. However, no ordering exists between the User and Resource sites, since neither of these two is assumed to be more trusted than the other in a comparable manner.

The Core site is a domain in which all core security and VO management services in XtreemOS are run. A detailed description of these services can be found in [29]. As a result, this domain constitutes the *root of trust* from which all other domains can be bootstrapped. Therefore, it is an important requirement for the domain to have the highest level of assurance in any XtreemOS-based Grid system.

The Core site may be split into two domains: The *Core site (offline)* and the *Core site (online)*, with the ordering *Core site (offline)* > *Core site (online)*. Essentially, the two domains represent two possible operation modes of the Core site. In the online mode, the domain is networked to other domains and so services and applications running remotely can access the domain and its core services. This has a lower level of assurance than in the offline case, although in practice, it is still a requirement to maintain high levels of assurance by adopting strong security protection measures.

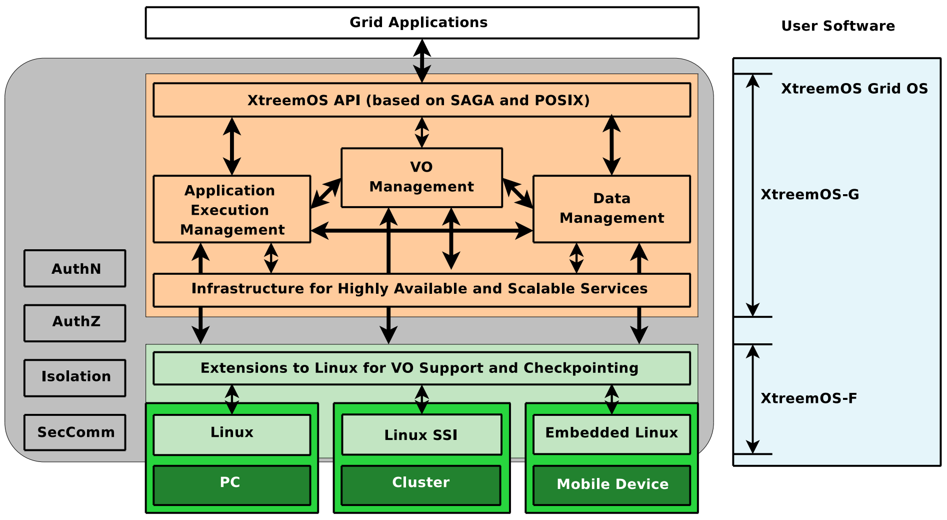
Grid Applications
User Software
XtreemOS API (based on SAGA and POSIX)
XtreemOS Grid OS
VO Management
Application Execution Management
Data Management
XtreemOS-G
AuthN
AuthZ
Isolation
SecComm
Infrastructure for Highly Available and Scalable Services
Extensions to Linux for VO Support and Checkpointing
XtreemOS-F
Linux
Linux SSI
Embedded Linux
PC
Cluster
Mobile Device

**Figure 1: XtreemOS Software Architecture [12].**

In the offline case, the domain has no network connections to any other domains, it utilises strict security measures and its services are only accessible via the interactions of authorised administrators. Therefore, the domain is considered to have the highest assurance level among all other domains.

The Resource site represents any domains in which resources (machines, services, software) are hosted and are connected to the Grid, therefore making them available to any VOs formed out of the Grid. The level of assurance of a resource site cannot be guaranteed. Finally, the User site is any domain hosting users of the Grid, which may apply to join VOs and avail of the VO resources. Like Resource sites, User sites have no guarantees regarding their assurance levels.

## 3.2 Elements of the Trust Model

We now turn our attention to the main elements constituting the XtreemOS trust model. These are shown in Figure 2, and can be classified into four main categories: *Certification Authorities*, *Credentials*, *Users* and *Resources*.

### 3.2.1 Certification Authorities

Certification authorities represent points of trust from which users and resources can obtain credentials to certify their identities and/or their attributes. XtreemOS defines three such authorities: the *Root Certification Authority* (Root CA), the *Credential Distribution Authority* (CDA) and the *Resource Certification Authority* (RCA). These authorities are organised in a hierarchy as shown in Figure 2, where trust is *delegated* from the Root CA to the CDA and then again to the RCA. This trust delegation implies that the CDA has its public key signed by the Root CA and that the RCA has its public key signed by the CDA. Therefore, any entity that trusts the Root CA will also trust the CDA and similarly then it will trust the RCA. The Root CA itself is a self-signing authority meaning that it will sign its own public key. An important advantage of adopting separate authorities for users (CDAs) and resources (RCAs) is that a clean separation of concerns could be achieved between users' and resources' credential management. This implies that the CDAs and the RCAs can easily be maintained (e.g. upgraded or exchanged) in an independent manner as long as the format of their certificates remains compatible with each other.

The Root CA is the *trust anchor* for any XtreemOS-based Grid system, which issues identity certificates to core XtreemOS services. The fact that the Root CA operates as the trust anchor means that it provides a point of reference to the system whenever other services are compromised and their certificates need to be reissued. Therefore, it is important to ensure that the Root CA is highly protected from unauthorised accesses and is running on a highly secured machine, hence it is usually operated in an offline mode. This means that any trust delegation (to the CDA) is carried out via offline means (e.g. emails, telephones, administrators' direct access using command line programs, etc.). No network connection is provided to any services or programs running on different machines. This is considered to provide a higher assurance level than if the Root CA provided online access, which would increase the risk of the Root CA being compromised (e.g. by the theft of the Root CA's private key).

The CDA is a subordinate of the Root CA; the Root CA delegates trust to the CDA. This delegation of trust means that the CDA can certify the public keys of users and core services such that any entity consuming the resulting certificates will be able to trace the chain of trust up to the Root CA. The user certificate that the CDA issues also contains the user's VO attributes, such as their VO membership. The CDA serves several purposes: First, it acts as the online certificate distribution frontend to the offline Root CA. Therefore, one can achieve a separation of concerns between the management of the Root CA's security and its online certificate distribution functionality. The CDA also permits the separation of functionality between user certification, and resource certification as performed by the RCA. To this end, the CDA delegates trust to the RCA by signing the RCA's public key. The CDA also acts as the authority enabling the formation of VOs out of users and RCAs who have been certified by a common CDA.

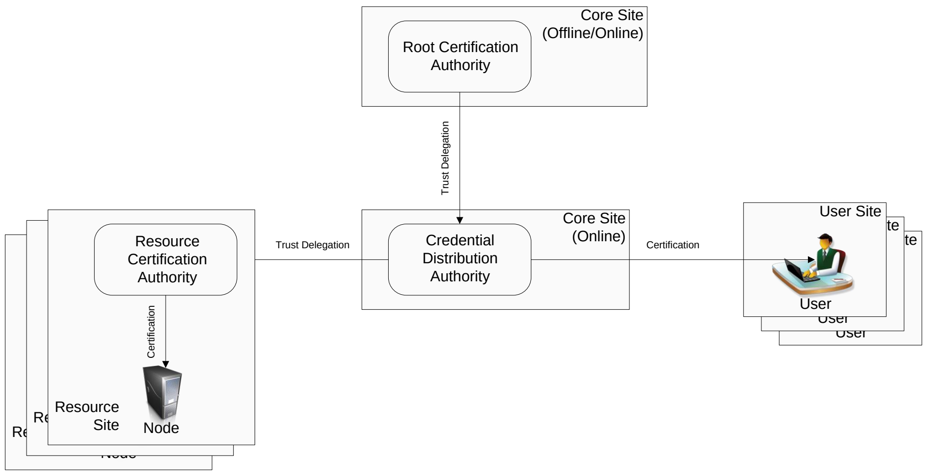
Core Site (Offline/Online)
Root Certification Authority
Trust Delegation
Core Site (Online)
Credential Distribution Authority
Trust Delegation
Resource Certification Authority
Certification
Resource Site
Node
Certification
User Site
User

**Figure 2: The XtreemOS Trust Model.**

Finally, the RCA is a subordinate of the CDA where the CDA delegates trust to each RCA for the purpose of managing resource certification in its resource domain. The RCA certifies the public keys of resources belonging to its site. Additionally, it also issues attribute certificates required for those resources, such as certificates stating the storage capacity, speed of processors and assurance and QoS levels of the resource. The RCA facilitates the management of the resource certification process within each resource site in the Grid. It also relieves the CDA from the task of certifying each individual resource in the Grid, which would require information about the resource only available to its local administrators.

### 3.2.2 Credentials

Digital credentials are pieces of data held by the different entities that provide some information about them. Credentials are usually cryptographic in the form of X.509v3 certificates [21]. We call XtreemOS user certificates simply *XOS-certificates*, certificates issued to XtreemOS services *service certificates*, and certificates issued by the RCA, *resource certificates*. Certificates in XtreemOS have the following two forms: *Identity* and *Attribute* certificates. Identity certificates are certificates that enable their consumer to cryptographically validate the binding between the identity of the certificate's holder and its public key. This binding is important as it allows the consumer in the future to validate the authenticity of any information signed by the certificate holder. In XtreemOS, identity certificates are issued to all entities in the Grid and therefore, they constitute the most essential trust mechanism without which entities cannot participate in VOs. On the other hand, attribute certificates are similar to the identity certificates except that they have an additional purpose of enumerating all the other attributes of the entity. Such attributes may include the role of the user in the VO or the computational power of the resource. In XtreemOS, attribute certificates carry the attributes in extension fields to the X.509v3 certificate format [21].

### 3.2.3 Users and Resources

Users are either humans or software that interact with the XtreemOS system and utilise the Grid resources within well-defined VOs. On the other hand, resources are the individual machines, or nodes, that offer services such as computational cycles and storage space to the Grid users. A resources is managed by a *resource administrator*, who could also be the *site administrator*.

## 3.3 Setting-up Trust

In XtreemOS, the above trust mechanisms are set-up using a number of processes, which lead to an *operational XtreemOS Grid* infrastructure out of which operational VOs can be formed. In the following paragraphs, we describe these processes in their order of their applicability.

### 3.3.1 Setting-up the Root CA certificate

The first process is to set-up the Root CA by running it in an offline non-networked machine, which is part of the Core site. The Grid administrator will run a command line program to generate the Root CA's private key and self-signed Root CA public key certificate. The Root CA's public certificate is then transferred to a networked machine for distribution to all nodes in this Grid. The Root CA will also create a certificate for the CDA.

### 3.3.2 Setting-up Core Services

The CDA creates service certificates for the core services. A CDA client program running on a core service generates the private key for that service and then uses it to sign a Certificate Signing Request (CSR), which contains the service's public key and other requested attributes, such as the type of the service and some descriptive text. The type of the service is defined by a constrained value representing any of the XtreemOS services and is checked by the service's clients during a Secure Sockets Layer (SSL) [14] handshake. There is an option for the CDA server to not automatically process all or any requests for service certificates. In this case, pending CSRs are stored in a special database for later manual processing by the Grid administrator, using the CDA's private key to create the service certificates. This option allows Grid administrators to apply different levels of trust to CSRs based on the identity of the requestor.

### 3.3.3 *The XtreemOS User Registration Process*

This process represents the entry point for users who wish to use the resources offered by a XtreemOS-enabled Grid. It is, in some sense, a pre-authentication step to what will follow in the next section. XtreemOS users will normally apply for an account in an XtreemOS Grid through a Web interface. This allows the applicant to enter their account details (username and password) and contact details (such as organisation and e-mail address). This process is similar to the vetting required to join an organisation. The manual vetting of applicants ensures that they are trustworthy to starting using this Grid. The Grid administrator has the option, when considering a registration request, of applying a level of scrutiny to the registration applications appropriate to the level of security and assurance required in their Grid. Once the user has had their application approved, they can use the Web interface to join existing VOs or to create their own VOs.

### 3.3.4 *The Resource Certificate Distribution Process*

During this process, an RCA, $R$, aims at obtaining a root certificate and an identity certificate from its associated CDA, $C$, through the following steps:

1. $[R \longrightarrow C]$: $CSR_R$
2. $[C \longrightarrow R]$: $(\langle cert_R \rangle_{SK_C}, \langle cert_C \rangle_{SK_C})$

where in the first step, $CSR_R$ is a request for certificate signing sent from the RCA to the CDA, $\langle cert_R \rangle_{SK_C}$ is the RCA's identity certificate signed by the private key of $C$, and $\langle cert_C \rangle_{SK_C}$ is a self-signed root certificate issued and signed by $C$. The communication is over an SSL connection [18] with mutual authentication. This means that communications are secret and authenticated in both directions. From now onwards, we use the notation $[A \longrightarrow Y]$ to denote an SSL-secured communication channel.

### 3.3.5 *Machine Certification by Local RCAs*

In general, machines need to register with at least one local RCA securely. Because machines are operated within the same administrative (trust) domain as their RCA, the problem of establishing a secure channel between a machine and its RCA is resolved locally within the domain. This will depend on the level of security and assurance adopted in the domain.

### 3.3.6 *Obtaining a User Certificate*

The final process is for the user to obtain their certificate from the CDA and start using XtreemOS Grid resources. This process is the focus of the next section, which introduces one of the many options that were considered in XtreemOS for authenticating users and enabling trust to be established between a user and a node belonging to an XtreemOS VO. It is also a critical part of the trust management process, since at this point, a user will be able to utilise resources belonging to other domains.

## 4. A USER/NODE MUTUAL AUTHENTICATION PROTOCOL

Several protocols for authentication were proposed in the context of establishing trust between XtreemOS users and nodes (resources), however of these only one was formally verified, which we discuss in this section, and which will serve us in the next section to demonstrate an example of the application of formal analysis tools to enhance the levels of trust assurance in complex operating systems.

The protocol is based on the classic Diffie-Hellman key agreement protocol [15]. At the end of this protocol, a shared secret key is agreed between communicating parties who (a) previously are unknown to each other; and (b) are under two different administrative domains, who may or may not use the same kind of authentication methods. This protocol aims to prove a user's identity in the context of a VO. The following is a list of notations used in the protocol:

- $U$: a user within a $VO$
- $User_{id}$: the user's unique identity within a $VO$
- $VO_{id}$: the identity of a $VO$ that $U$ is registered with
- $N$: a resource node
- $VOM$: a VO management authority that runs a CDA
- $g$, $n$: the Diffie-Hellman (DH) parameters, where $g$ is the DH exponent and $n$ is the size of the DH field which the computation is based on.
- $R_Y$: a random number generated by entity $Y$
- $G_Y$: a constant where $G_Y = g^{R_Y} \ mod\ n$
- $G_{YX}$: a constant where $G_{YX} = (g^{R_Y} mod\ n)^{R_X} mod\ n$
- $T_Y$: a timestamp generated by entity $Y$
- $K_Y^r$: the private key of entity $Y$
- $K_Y^u$: the public key of entity $Y$
- $\langle M \rangle K_Y^r$: a message $M$ signed by $Y$'s private key
- $\{M\} K_Y^u$: a message $M$ encrypted by $Y$'s public key
- $RndMsg_Y$: a random message generated by entity $Y$

The mutual authentication protocol consists of the following messages described in the classical Alice-Bob style:

1. $[U \rightarrow VOM]$: $User_{id}$, $VO_{id}$, $g$, $n$, $G_U$
2. $[VOM \rightarrow U]$: $\langle User_{id}, VO_{id}, g, n, G_U, T_{VOM} \rangle\ K_{VOM}^r$
3. $U \rightarrow N$: $\{\ \langle User_{id},\ VO_{id},\ g,\ n,\ G_U,\ T_{VOM} \rangle K_{VOM}^r$, $RndMsg_U$, $T_U\ \} K_N^u$
4. $N \rightarrow U$: $G_N$, $\{RndMsg_U, T_U\} G_{UN}$, $\{RndMsg_N, T_N\} G_{UN}$
5. $U \rightarrow N$: $\{RndMsg_N, T_N\} G_{NU}$

The protocol commences when the user, $U$, in (1.) requests from its home VO management authority, $VOM$, an XOS certificate by submitting to VOM its user identity $U_{id}$, the id of the VO it belongs to, $V_{id}$, the Diffie-Hellman parameters, $g, n$, and the constant $G_U$, which the user computes based on a fresh random number, $R_U$, it generated. This initial message is assumed to be communicated over a mutual SSL-secured channel shared between $U$ and $VOM$, therefore it is not possible for $U$ to masquarade itself as someone else to the $VOM$ server. Once $VOM$ receives the message, it will check the validity of $U$'s membership in the VO.

In the next message (2.), $VOM$ replies to $U$ also on the same SSL connection by sending it an XOS certificate signed by its private key, $K_{VOM}^{r}$, and carrying a timestamp, $T_{VOM}$, denoting the expiry time of the certificate. $U$ is then able to use the certificate within its time validity to authenticate itself to any node in the VO that trusts $VOM$ and to establish a shared session with that node without. Up until now, all communications have been assumed to run over an SSL-secured connection. Since nodes can have varying security communications capabilities, no such assumption is made in the following messages between the node and the user.

In message (3.), $U$ contacts one such node, $N$, over an insecure public channel. $U$ sends to $N$ the certificate it received from $VOM$ along with a timestamped message, $RndMsg_U, T_U$, all encrypted with the public key of the node, $K_N^u$. This ensures that the message is fresh and that it can only be decrypted by $N$. Once $N$ receives the message, it checks the validity of the certificate and if it is valid, it then generates the public constant, $G_N$, and the session key $G_{UN}$.

In message (4.), $N$ then sends to $U$ the public constant $G_N$ along with the original timestamped message of $U$ encrypted under the session key $G_{UN}$ and a new timestamped message, $RndMsg_N, T_N$ generated by $N$ and encrypted with $G_{UN}$. Upon the receipt of this message, $U$ generates its own copy of the session key, $G_{NU}$, which is equivalent to $G_{UN}$. $U$ then uses $G_{NU}$ to decrypt the two parts of the message containing $RndMsg_U, T_U$ and $RndMsg_N, T_N$. If $RndMsg_U, T_U$ is the same as the original and $RndMsg_N$ is fresh (i.e. $T_N$ is recent), then $N$ is authenticated and $G_{NU}$ is accepted as the session key by $U$. Finally, in message (5.), $U$ sends to $N$ a message, $RndMsg_N, T_N$, encrypted with its session key $G_{NU}$. $N$ then receives this message, decrypts it, and then checks that the pair $RndMsg_N, T_N$ is the same as the original one generated by $N$. If this is the case, $N$ accepts the authenticity of $U$ and the use of $G_{UN}$ as the session key. At the end of this step, both $U$ and $N$ will have authenticated themselves and accepted $G_{NU} = G_{UN}$ as their session key.

## 5. FORMAL MODEL AND ANALYSIS OF THE PROTOCOL

In this section, we present the formal model and analysis of the mutual authentication protocol between XtreemOS users and nodes. The approach we followed was to first define what we meant by mutual authentication. Then we constructed a model of the protocol in a formal language that is expressive enough to be able to capture concepts and mechanisms used in the protocol and that is supported by automated verification tools. Finally, we used the verification tools to verify that the mutual authentication property claimed by the protocol is indeed upheld in the face of external attackers.

Our choice of approach was mainly driven by the following considerations, some of which were the result of requirements and constraints imposed by the XtreemOS project:

- The natural suitability of the input language of the tool for modelling security protocols. Such a language has to be message-passing and capable of expressing security primitives (such as cryptographic functions).
- The presence of supporting tools and the degree of automation of those tools as well as their efficiency and natural ability in verifying security properties.
- The approach as well as the supporting tools must be able to deal with infinite runs of the system, since security protocols are assumed to have an infinite number of sessions.

After considering the above different approaches, our conclusion was that abstract interpretation using process algebra such as the $\pi$-calculus [22] was very well suited to the modelling of security protocols. Furthermore, this approach is supported by one of the most efficient static analysis tools targeted at the analysis of security protocols, namely ProVerif [7].

### 5.1 A Formal Definition of Mutual Authentication

Our definition of authentication is based on the type (3) authentication as specified by Lowe [26, §3.3]. In this type, two entities, $a$ and $b$ can mutually authenticate themselves and agree on additional information specific to the protocol session if each is convinced that the other entity has participated in the run and that the information exchanged is the same. This is achieved using the concept of *commit* and *running* events. The commit-running events provide a mechanism to ensure the temporal ordering of protocol steps in a manner leading to (mutual) authentication. The occurrence of a commit event in $b$ must imply that the running event in $a$ has already occurred. This means that $b$ has authenticated $a$. The opposite authenticates $b$ to $a$. The commit event must happen after the end of the protocol, whereas the running event can happen at anytime during the protocol but must happen before it is over. More formally, it is possible to define one-way authentication as follows.

DEFINITION 1 (ONE-WAY AUTHENTICATION). *Given processes, $A$ and $B$, we say that $A$* authenticates $B$ agreeing on *$M$ if $A$ executes event $commit_A(M)$ and $B$ executes event $running_B(M')$ and the following is true:*

$$(commit_A(M) \Rightarrow running_B(M')) \ \wedge \ (M = M')$$

In other words, the definition of one-way authentication states that the occurrence of $running_B(M')$ precedes the occurrence of $commit_A(M)$. Furthermore, by the time both these events have occurred, their corresponding parameters, $M$, $M'$, must be the same according to some definition of the = relation.

Now, the definition of mutual authentication is formalised as follows, based on the definition of one-way authentication.

DEFINITION 2 (MUTUAL AUTHENTICATION). *We say that $A$ and $B$* mutually authenticate each other*, if the following holds true:*

$A$ authenticates $B$ agreeing on $M$ $\Leftrightarrow$
$B$ authenticates $A$ agreeing on $M'$

Here, it is not necessary that $M = M'$, however, in session key agreement protocols where the aim is to establish a common session key, the two values must agree if they represent the common session key. In fact, in our analysis to follow, this will be the case.

## 5.2 The Applied-$\pi$ Calculus Language

The syntax of the input language is given in Figure 3. This syntax is based largely on a version of the $\pi$-calculus

| | | |
|---|---|---|
| ⟨*term*⟩ | ::= | ⟨*ident*⟩ \| `(`*seq*⟨*term*⟩`)` |
| | | \| ⟨*ident*⟩`(`*seq*⟨*term*⟩`)` |
| ⟨*fact*⟩ | ::= | ⟨*ident*⟩`:`*seq*⟨*term*⟩ |
| | | \| ⟨*term*⟩ `<>` ⟨*term*⟩ |
| | | \| ⟨*term*⟩ `=` ⟨*term*⟩ |
| ⟨*process*⟩ | ::= | `(` ⟨*process*⟩ `)` |
| | \| | `!` ⟨*process*⟩ |
| | \| | `0` |
| | \| | `new` ⟨*ident*⟩`;` ⟨*process*⟩ |
| | \| | `if` ⟨*fact*⟩ `then` ⟨*process*⟩ [`else` ⟨*process*⟩] |
| | \| | `in(`⟨*term*⟩`,` ⟨*term*⟩`)`[`;` ⟨*process*⟩] |
| | \| | `out(`⟨*term*⟩`,` ⟨*term*⟩`)`[`;` ⟨*process*⟩] |
| | \| | `let` ⟨*term*⟩ `=` ⟨*term*⟩ `in` ⟨*process*⟩ [`else` ⟨*process*⟩] |
| | \| | ⟨*process*⟩ \| ⟨*process*⟩ |
| | \| | `event` ⟨*term*⟩[`;` ⟨*process*⟩] |

**Figure 3: The syntax of processes in the applied $\pi$-calclulus.**

[22] called the applied $\pi$-calculus [2], which extends the $\pi$-calculus with functional constructors/destructors and equational theories defining how constructors and destructors are related to each other. The meaning of the syntax is described informally as follows: A term, ⟨term⟩, is either an identifier, a sequence of terms or the application of a function to a sequence of terms. A fact, ⟨*fact*⟩, is either a predicate applied to a sequence of terms, an inequality check of two terms or an equality check of two terms. Using terms and facts, a process is then defined according to the following constructs:

- `(` ⟨*process*⟩ `)`: a process enclosed by two brackets to remove ambiguity.
- `!` ⟨*process*⟩: a replicated process, which is capable of spawning as many copies of itself as is required by the context.
- `0`: the null process, which is incapable of any behaviour.
- `new` ⟨*ident*⟩`;` ⟨*process*⟩: the process that creates a new identifier with scope restricted to the residual process.
- `if` ⟨*fact*⟩ `then` ⟨*process*⟩ [`else` ⟨*process*⟩]: a process that checks whether the specified fact is true. If so, it chooses the `then`-branch. Otherwise, it proceeds as the `else`-branch.
- `in(`⟨*term*⟩`,` ⟨*term*⟩`)`[`;` ⟨*process*⟩]: an input process that receives a term over a channel name (the first indicated term) and uses that term to replace its input parameter (the second indicated term). It then proceeds as the residual process. The input parameter has a scope restricted to the residual process.
- `out(`⟨*term*⟩`,` ⟨*term*⟩`)`[`;` ⟨*process*⟩]: an output process, which sends over a channel (the first indicated term) a message (the second indicated term) and then proceeds as the residual process.
- `let` ⟨*term*⟩ `=` ⟨*term*⟩ `in` ⟨*process*⟩ [`else` ⟨*process*⟩]: a let-process, which assigns a term (the second indicated) to another (the first indicated) with the scope of the residual process. If the assignment fails, the let-process proceeds as the else-process indicated at the end.
- ⟨*process*⟩ | ⟨*process*⟩: the parallel composition of two processes, which has an interleaving semantics.
- `event` ⟨*term*⟩[`;` ⟨*process*⟩]: an event that has a name, a possible sequence of terms that it may synchronise on and is followed by the residual process.

For a formal semantics of the above syntax, we refer the reader to [7] and [2].

## 5.3 The Formal Model of the Protocol

We define in Figure 4 a model of the mutual authentication protocol as described in Section 4 using the syntax of the ProVerif tool. The protocol definition consists of three process definitions: *vom* representing the VO management process, $p0$ representing the user process and $p1$ representing the node process. The protocol definition starts creating the private and public parts of the *vom* process and initialises it with the private part of the key. It also advertises the public part over a public channel, *attc*, that can be read by the attacker. It then starts initialising any number of nodes with their private keys and with the public key of *vom* as well as advertising the public key of these nodes over *attc*. At the same time, the protocol process runs the node, the user and vom processes in parallel with each other.

The *vom* process is ready to accept a request from a user for an XOS certificate over a channel, *vom*, known only to the vom and user processes. The request is dealt with by sending to the user a signed certificate and then signaling using an event the termination of vom. The assumption here is that there is a single user willing to communicate with several nodes. The user then, after receiving the certificate, starts a replicated process which is able to start mutually authenticating as many nodes as available. Note the presence of the commit-running events for both the user and node processes, which will be used to verify the success of authentication. The specification of the protocol utilises the following equations defining the relationship among functions:

$$getmess(sign(m,k)) = m \tag{1}$$
$$checksign(sign(m,k), pk(k)) = m \tag{2}$$
$$pubdec(pubenc(x, pk(y)), y) = x \tag{3}$$
$$dec(enc(x,y), y) = x \tag{4}$$
$$f(g(y), x) = f(g(x), y) \tag{5}$$

where equations 1-3 define public-key cryptography operations, equation 4 defines secret-key cryptography and equation 5 defines the Diffie-Hellman pair of functions. This representation of the Diffie-Hellman functions in this manner is popular in process-algebraic-based protocol definitions [1, 7] and it's motivated by the analysis algorithms. Essentially, $g(R_Y) = G_Y$ and $f(G_Y, R_X) = G_{YX}$ in terms of the $G_Y$ and $G_{YX}$ defined in Section 4. The user process, however, also passes to the vom process the Diffie-Hellman parameters, $DHexp$ and $DHfld$, as numbers representing the exponent and the field size. This is included to model faithfully the protocol description.

```
(* The VOM Process Definition *)
let vom =
    (* Get initialised with the VOM private key *)
    in(tvom, skVOM0);
    (* Receive a Xcert request from a user *)
    in(vom, (v1, v2, v3, v4, v5));
    (* Check that the user's name is u - (identification abstraction) *)
    if v1=u then
    (* Send user's Xcert signed by VOM's private key and timestamped *)
    out(vom, sign((v1, v2, v3, v4, v5, Tvom), skVOM0))

(* The User Process Definition *)
let p0 =
    (* Create a new user random number *)
    new Ru;
    (* Request an XCert from the VOM *)
    out(vom, (u, vid, DHexp, DHfld, g(Ru)));
    in(vom, xcert);
    !(
    (* Get initialised with a node's name and public key *)
    in(tu, (nv, pkN0));
    (* Create a new message *)
    new MSGu;
    (* Contact the node and wait for the response *)
    out(nv, pubenc((xcert, MSGu, Tu), pkN0));
    in(u, (m1, m2, m3));
    (* Generate the session key *)
    let ku = f(m1, Ru) in
    (* Decrypt the node messages using the session key *)
    let m4=dec(m2, ku) in
    let m5=dec(m3, ku) in
    (* Check that the message returned by the node is the same one
       sent by the user *)
    (* If so, signal a running event *)
    if m4 = (MSGu, Tu) then event p0running(f(g(Rn), Ru));
    (* Send to the node its own message back and
    signal a commit event agreeing on the session key *)
    out(nv, m3);event p0commit(f(g(Rn), Ru)))

(* The Node Process Definition *)
let p1 = !(
   (* Initialise the node with its private key and the VOM public key *)
      in(tn, (skN1, pkVOM1));
   (* Receive a request from the user *)
      in(nv, x);
   (* Decrypt message sent by user using the node's private key *)
      let (x0, x1, x11) = pubdec(x, skN1) in
   (* Verify XCert signed by VOM using the latter's public key *)
      let (x2, x3, x4, x5, x55, x555) = checksign(x0, pkVOM1) in
   (* Check that VO id is correct and that timestamp is fresh *)
      if x3 = vid then if x4 = DHexp then
      if x5 = DHfld then if x555 = Tvom then
   (* Create a new random number and message *)
      new Rn; new MSGn;
   (* Create the session key signal the running event agreeing on
      the session key *)
      let kn = f(x55, Rn) in event p1running(f(g(Ru), Rn));
   (* Send message to the user including g(Rn) *)
      out(x2, (g(Rn), enc((x1, x11), kn), enc((MSGn, Tn), kn)));
   (* Receive the reply from the user *)
      in (nv, x6);
   (* Decrypt the message *)
      let x7 = dec(x6, kn) in
   (* Check that received message is the same as original one sent *)
   (* If so, signal the commit event agreeing on the session key *)
      if x7 = MSGn then event p1commit(f(g(Ru), Rn)))

(* The Protocol Definition *)
   (* Create secure private channel names *)
new tvom; new vom; new tn; new tu; (
   (* Create private/public key pair for VOM *)
new skVOM; let pkVOM = pk(skVOM) in
   (* Initialise VOM with its private key and advertise its public key *)
out(tvom, skVOM); out(attc, pkVOM);
   (* Initialise as many nodes with their private key and VOM's public key *)
   (* Advertise the node's public key and initialise the user with
      the node's public key *)
!(new skN;
let pkN = pk(skN) in out(attc, pkN);
(out(tn, (skN, pkVOM)) | out(tu, (n, pkN)))))
   (* Run the node, the user and the VOM process concurrently *)
   (* The attacker is listening on channel attc *)
| (p1) | (p0) | (vom) | (!(in(attc, v)))
```

**Figure 4: The applied $\pi$-calculus model of the XtreemOS mutual authentication protocol.**

## 5.4 The Data Leakage Analysis

The first analysis of the XtreemOS mutual authentication protocol using the ProVerif tool was carried out to determine which data terms can possibly be leaked to the external attacker. The results of the analysis are given in Figure 5.

**Terms Leaked**
*pkN*, *pkVOM*, *DHexp*, *DHfld*, *u*, *vid*, *nv*, *attc*, *g*(*Rn*)

**Terms Not Leaked**
*Ru*, *Rn*, $f(g(Ru), Rn)$, $f(g(Rn), Ru)$, $g(Ru)$, *MSGu*, *MSGn*, *skN*, *skVOM*, *Tu*, *Tn*, *Tvom*, *tu*, *tn*, *vom*, *tvom*

**Figure 5: Leakage analysis for the XtreemOS mutual authentication protocol.**

From these results, it is clear that the attacker was not able to construct or capture the session key, $f(g(Ru), Rn)$, which is the same as $f(g(Rn), Ru)$ nor indeed any of the terms that need to be kept secret such as the random numbers generated by the user and the node, $R_u$, $R_n$, the private keys of the node and VOM, *skN*, *skVOM*, and the exchanged messages generated by the user and the node, *MSGu*, *MSGn*.

On the other hand, the analysis reveals that the attacker is able to capture the term, $g(Rn)$, which is a representation of $G_N$ in the protocol's Alice-Bob description of Section 4 and which is sent in the clear by the node back to the user. This implies that the attacker may be able to guess the value of the random number *Rn* if the node did not choose a sufficiently large *Rn* (in the order of 100 digits long). Similarly, the attacker may also guess the value of the session key $f(g(Rn), Ru)$ given *Rn*, if the user did not manage to choose a sufficiently large random number *Ru*. Apart from that, the attacker's knowledge is confined to the public knowledge of the names *pkN*, *pkVOM*, *DHexp*, *DHfld*, *u*, *vid*, *nv* and *attc*.

We note that the attacker is unable to capture $g(Ru)$ generated by the user, even though this term is sent on the clear initially to the VOM. This is due to the fact that the *vom* channel is declared as a secure channel using the restriction (`new` *vom*) in the definition of the protocol. Even though in the model, such channels are possible to describe, in reality, one cannot assume that there are channels secure by their nature. Therefore, our model is somehow unrealistic in that it assumes the existence of such naturally secure channels. One alternative to move away from this assumption would be to use encryption for initial communication between the use and VOM. In that case, it would be possible to send the message (including $g(Ru)$) over an insecure public channel.

## 5.5 The Mutual Authentication Analysis

For the second analysis, we wanted to verify the mutual authentication property for the protocol using commit-running events. The ProVerif tool was able to prove that the following two implications are true in the case of passive attackers:

$$\mathit{p1commit}(f(x55, Rn)) \Rightarrow \mathit{p0running}(f(m1, Ru)) \quad (6)$$
$$\mathit{p0commit}(f(m1, Ru)) \Rightarrow \mathit{p1running}(f(x55, Rn)) \quad (7)$$

Where $x55$ is a term instantiated to $g(Ru)$ and $m1$ is a term instantiated to $g(Rn)$. According to the equation $f(g(x), y) = f(g(y), x)$, we can infer that $f(x55, Rn) = f(m1, Ru)$ in both cases, therefore we can say that the above result satisfies the definition of mutual authentication (Section 5.1) for both the node and the user in the case of passive attackers.

However, when running the tool for the case of an active attacker (e.g. Dolev-Yao [16]), we found that (6) does not hold. This is simply becuse it is possible for $U$ to abort once $N$ sends message 4, and this message is then intercepted by the attacker who can *replay* the second part of message 4 back to $N$. Since $G_{UN} = G_{NU}$, $N$ will incorrectly conclude that $U$ is still alive (and authentic), therefore violating the definition of (6) above.

## 6. DISCUSSION ON THE PROTOCOL VERIFICATION

Technically, we found that the modelling of the XtreemOS mutual authentication protocol was naturally straightforward in the message-passing process algebra due to the similarity in the nature of the protocol and the process algebraic language. The existence of an automatic and efficient verification tool backed up by a formal theory (in the style of Horn clauses [20]) facilitated the task of verifying security properties such as data leakage and mutual authentication related to our protocol. Both the data leakage and the mutual authentication analyses revealed expected results for the case of passive attackers. One recommendation is related to the fact that some of the data terms, such as $g(Rn)$, are sent on the clear and captured by the attacker. This requires that the user and the node must choose large random numbers, *Rn*, *Ru*, in order to prevent the attacker from guessing the session key. This is a well-known requirement in Diffie-Hellman-based protocols that reveal some of the protocol-generated data on the clear. In the case of active attackers, authentication of users is not possible due to replay attacks, which is also expected as a result of the communication of some messages in the clear.

Since security protocols are often specified using informal Alice/Bob-style notation (as was the case in the original specification of the protocol in Section 4), and despite the fact that this is usually sufficient for giving a reasonable description of the exchange of messages between the different agents involved in the protocol, it does not provide any detail or formal grounds for reasoning about the internal behaviour of those agents. Such internal behaviour is necessary to determine whether the agent maintains its protocol obligations, such as non-leakage of session keys and the verification of digital signatures. The modelling exercise here provided necessary detail in the protocol's specification, which was missing from the original Alice/Bob-style specification to be able to reason about the protocol's properties using formal analysis tools.

## 7. CONCLUSION AND FUTURE WORK

We presented in this paper a case study in the application of formal modelling and verification techniques using process algebra and associated tools to enhance the assurance of a mutual authentication protocol, part of the trust model for a large-scale distributed operating system for Grids called XtreemOS. The modelling itself enabled the protocol as well as the trust model to be better specified thus providing valuable guidance for the later design stage of the XtreemOS system. We plan to utilise the expertise gained in this exercise to model and verify future protocols in the context of complex distributed systems.

## 8. REFERENCES

[1] M. Abadi, B. Blanchet, and C. Fournet. Just fast keying in the pi calculus. In D. Schmidt, editor, *Programming Languages and Systems: Proceedings of the* 13th *European Symposium on Programming*, volume 2986 of *Lecture Notes in Computer Science*, pages 340–354, Barcelona, Spain, Mar. 2004. Springer Verlag.

[2] M. Abadi and C. Fournet. Mobile Values, New Names, and Secure Communication. In *Proceedings of the* 28th *ACM Symposium on Principles of Programming Languages*, pages 104–115, London, UK, Jan. 2001. ACM Press.

[3] A. Arenas, B. Aziz, J. Bicarregui, B. Matthews, and E. Y. Yang. Modelling security properties in a grid-based operating system with anti-goals. In *Proceedings of the 2008 Third International Conference on Availability, Reliability and Security*, pages 1429–1436, Washington, DC, USA, 2008. IEEE Computer Society.

[4] B. Aziz, D. Gray, G. Hamilton, F. Oehl, J. Power, and D. Sinclair. Implementing Protocol Verification for E-Commerce. In *Proceedings of the* 2nd *International Conference on Advances in Infrastructure for E-Business, E-Science and E-Education on the Internet*, L'Aquila, Italy, Aug. 2001.

[5] B. Aziz and G. Hamilton. Verifying a delegation protocol for grid systems. *Future Generation Computer Systems*, 27(5):476 – 485, 2011.

[6] B. Aziz, G. Hamilton, and D. Gray. A static analysis of cryptographic processes: The denotational approach. *Journal of Logic and Algebraic Programming*, 64(2):285–320, Aug. 2005.

[7] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of the* 14th *IEEE Computer Security Foundations Workshop*, pages 82–96, Cape Breton, Nova Scotia, Canada, July 2001. IEEE.

[8] C. Bodei, P. Dagano, F. Nielson, and H. R. Nielson. Static analysis for the $\pi$-calculus with applications to security. *Information and Computation*, 168(1):68–92, July 2001.

[9] X. Consortium. Design of the architecture for application execution management in xtreemos. In *XtreemOS public deliverables - D3.3.2.* Work Package 3.3, May 2007.

[10] X. Consortium. Methodology and design alternatives for trust services in xtreemos. In *XtreemOS public deliverables - D3.5.11.* Work Package 3.5, February 2010.

[11] M. Coppola, Y. Jégou, B. Matthews, C. Morin, L. P. Prieto, O. D. Sánchez, E. Yang, and H. Yu. Virtual Organization Support within a Grid-wide Operating System. *IEEE Internet Computing*, 12(2):20–28, March 2008.

[12] T. Cortes, C. Franke, Y. Jégou, T. Kielmann, D. Laforenza, B. Matthews, C. Morin, L. P. Prieto, and A. Reinefeld. XtreemOS: a Vision for a Grid Operating System. XtreemOS Technical Report # 4, May 2008.

[13] D. Denning. A lattice model of secure information flow. *ACM Transactions on Programming Languages and Systems*, 19(5):236–243, May 1976.

[14] T. Dierks and E. Rescorla. Rfc 5246: The transport layer security (tls) protocol version 1.2, Aug. 2008.

[15] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, Nov. 1976.

[16] D. Dolev and A. Yao. On the security of public key protocols. In *Proceedings of the* 22nd *Annual Symposium on Foundations of Computer Science*, pages 350–357, Oct. 1981.

[17] J. Feret. Confidentiality analysis of mobile systems. In *Proceedings of the* 7th *International Static Analysis Symposium*, volume 1824 of *Lecture Notes in Computer Science*, pages 135–154, University of California, Santa Barbara, USA, June 2000. Springer Verlag.

[18] A. O. Freier, P. Karlton, and P. C. Kocher. The ssl protocol version 3.0, Nov. 1996.

[19] T. Grandison and M. Sloman. A Survey of Trust in Internet Applications. *IEEE Communications Surveys and Tutorials*, 3(4), September 2000.

[20] A. Horn. On sentences which are true of direct unions of algebras. *The Journal of Symbolic Logic*, 16(1):14–21, Mar. 1951.

[21] R. Housley, W. Polk, W. Ford, and D. Solo. Rfc 3280 - internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile, April 2002.

[22] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (parts I & II). *Information and Computation*, 100(1):1–77, Sept. 1992.

[23] C. Morin. XtreemOS: A Grid Operating System Making your Computer Ready for Participating in Virtual Organizations. In *Proceedings of the Tenth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2007)*, pages 393–402. IEEE Computer Society, 2007.

[24] F. Nielson, R. R. Hansen, and H. R. Nielson. Abstract interpretation of mobile ambients. *Science of Computer Programming*, 47(2–3):145–175, May 2003.

[25] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis.* Springer-Verlag, 1999.

[26] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *Modelling and Analysis of Security Protocols.* Addison-Wesley Professional, Dec. 2000.

[27] P. Ryan and S. Schnieder. *Modelling and Analysis of Security Protocols.* Addison-Weslley, 2001.

[28] XtreemOS Consortium. Requirement documentation and architecture for xtreemfs. In F. Hupfeld, editor, *XtreemOS public deliverables - D3.4.1.* Work Package 3.4, November 2006.

[29] XtreemOS Consortium. Fourth specification, design and architecture of the security and vo management services. In *XtreemOS public deliverables - D3.5.13.* Work Package 3.5, Dec. 2009.