Securing Critical Infrastructures, Grenoble, October 2004

SECURE ARCHITECTURES FOR CLUSTERS AND GRIDS

Sébastien Varrette ($\star$), Jean-Louis Roch($\star\star$), Yves Denneulin ($\star\star$), Franck Leprévost ($\star$)
($\star$) Université du Luxembourg, Faculté de Droit, Economie et Finance, Luxembourg
($\star\star$) Projet APACHE (CNRS/INPG/INRIA/UJF), Laboratoire ID-IMAG, Grenoble, France

I. Introduction

For applications like multi-physics simulations or complex data analysis, todays needs in computations require to gather thousands of computers geographically scattered and interconnected throw the Internet. Also the use of large scale global computing platforms – from a grid that couples several clusters of computers to peer-to-peer systems – has been experimented for some compute intensive high-end applications, such as the popular Seti@home [3] or BlueGene [1].

However, extending such global computing platforms to a wide class of applications and resources faces several critical security issues concerning the software architecture that manages the grid: [9]:
- users and machines have to be authenticated;
- as regards communications, privacy, integrity and non-repudiation are still basic requirements;
- component failures and disconnections are frequent events: the system has to ensure fault-tolerance for the application;
- the results computed on remote resources, that may be victims of Trojan horses, have to be certified.

In this paper, we firstly compare and classify (§II) the various security policies that have been developed for clusters and grids, from point-to-point security to private key (Kerberos, Kryptoknight) and public key (PKI) infrastructures. Coupling several clusters requires compliance with the local security policies on each local cluster, either by deploying a virtual private network (VPN) or based on a PKI infrastructure (Globus [8], DataGrid [4]). In order to resist to attacks by Trojan horses, output results are checked on the replication of computations, either total replication [19] or, more recently, partial replication [13].

Yet, tackling both security issues in a global architecture remains an open problem. In section IV, we propose a security infrastructure that address both problems. Smart cards (§III) are used in order to address authentication issues while using the system from a non trusted machine.

II. Classification of security technologies on the grid

A. Secure architecture for a local cluster

A.1 Point-to-point channel solutions

SSH is often seen as a way to improve security in networks. But how can it be used in a cluster environment?

The idea is there to have either a server `sshd` and a client `ssh` on each nodes of the cluster. To provide Single Sign On[1] the files `authorized_keys` must include the public keys for each users/machines and must be placed on each nodes of the cluster. Then, thanks to SSH agent, secure authentication of the users and the machines, privacy and integrity of the communications and Single Sign On can be automatically provided on the cluster.

This method has been practically experienced as it is currently one of the possible alternative in the settlement of a Taktuk network which is used in Inuktitut [14].

Yet, this first approach has many drawbacks:
- initialization and modifications in the users accounts must be done on each node;
- administration work to maintain a running system (for instance to add new users etc...) is time expensive.

A.2 Kerberos

Kerberos [18] is an authentication system developed by the MIT that used a centralized universally trusted authority called the KDC[2]. It is based on private key cryptography[3]. Kerberos shares with each entities u in the network a secret key K_u and the knowledge of this key is assumed as a proof of identity.

Notations used in Kerberos are summarized in table.I.

a	Alice
b	Bob
id_u	public information that identify u (ex : name, @IP)
t	Time of request
t_{end}	expiration time of the ticket
K_u	secret key of u
$K_{u,v}$	session key between u and v
$K\{x\}$	encryption of text x using key K
$T_{u,v}$	Encrypted ticket for u to use v
$A_{u,v}$	Encrypted Authentificator of u for v

TABLE I
Notations used in Kerberos

In Kerberos, a client c (generally either a user or a service) may obtain a *ticket* for a given service s from the KDC.

A ticket $T_{c,s}$ is a temporary credential that allows s to safely check the identity of the owner c of the ticket to which it has been delivered. In practice, $T_{c,s} = s, K_s\{id_c, t, t_{end}, K_{c,s}\}$. It contains the name of the service (s) that c wants to use and a set of informations encrypted with the service's private key K_s, particularly the identity of c and a session key $K_{c,s}$. Since only s and the KDC share the private key K_s, the ticket is known to be authentic. $K_{c,s}$ is securely shared between c and s (c received it encrypted with its own private key K_c so that only him

[1]users will authenticate - i.e enter their password - only once a day typically

[2]Key Distribution Center

[3]but there are possible extensions based on certificate and therefore on public key cryptography. See [20]

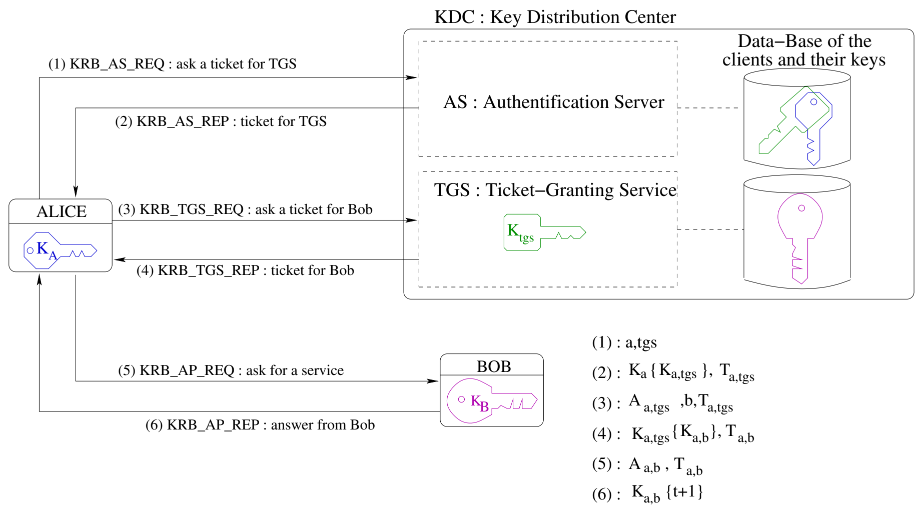
KDC : Key Distribution Center
Data−Base of the clients and their keys
AS : Authentification Server
TGS : Ticket−Granting Service
K_{tgs}
(1) KRB_AS_REQ : ask a ticket for TGS
(2) KRB_AS_REP : ticket for TGS
(3) KRB_TGS_REQ : ask a ticket for Bob
(4) KRB_TGS_REP : ticket for Bob
(5) KRB_AP_REQ : ask for a service
(6) KRB_AP_REP : answer from Bob
ALICE
K_A
BOB
K_B
(1) : a,tgs
(2) : $K_a\{K_{a,tgs}\}, T_{a,tgs}$
(3) : $A_{a,tgs}$,b,$T_{a,tgs}$
(4) : $K_{a,tgs}\{K_{a,b}\}, T_{a,b}$
(5) : $A_{a,b}$, $T_{a,b}$
(6) : $K_{a,b}$ {t+1}

Fig. 1. Authentication protocol in Kerberos

can decrypt it). The session key will be used to encrypt further communications and for *authenticators*.

An authenticator $A_{c,s}$ is another type of credential used in Kerberos. It is delivered by c with $T_{c,s}$ and $A_{c,s} = E_{K_{c,s}}(id_c, t)$. Thus, $A_{c,s}$ serves two purposes :

- it proves that c knows the shared session key $K_{c,s}$;
- c can then surely provide his identity to s, and s can safely check it thank to the ticket $T_{c,s}$.

A brief overview of the Kerberos protocol is exhibited in fig.1. More details can be found in [18]. An architecture based on Kerberos in a cluster has many advantages:

- Kerberos negotiates secure authentication and optionally encrypted communications between two nodes;
- no password are transmitted over the network;
- Kerberos provides Single Sign On;
- authentication is centrally managed;
- this is an IETF standard supported by numerous OS.

Yet, there are also various drawbacks [5]:

- Each credential encloses a time-stamp. Consequently, every nodes in the cluster have to be synchronized (for instance with NTP[4] or SNTP[5]). Yet these protocols are often not secure.
- The system is centralized. The KDC is then a privileged point for attacks like Denial of Service. It is also a limiting factor for performances as every node has to access it. Yet, there are possibilities to duplicate the KDC.

As regards typical attacks on network architectures, Kerberos is still vulnerable to ”guessing password” attacks. Replay attacks are limited thanks to the use of time-stamps. Trojan horses on nodes are still not avoided.

Yet, despite those drawbacks, a Kerberos architecture is a good way to secure a given cluster. But it should not be extended for the case of grids.

[4]Network Time Protocol
[5]Simple Network Time Protocol

A.3 KryptoKnight

KryptoKnight [17] is a Kerberos-like architecture developed by IBM. It is an authentication and key-distribution system based on MAC -*Message Authentication Code*. It provides four security services:

1. Single Sign On;
2. user and mutual authentication;
3. key distribution;
4. authentication of origin and content of data.

This last service is a first difference with Kerberos. In addition:
1. KryptoKnight uses a hash function for authentication and encrypting tickets.
2. It does not rely on synchronized clocks; it uses nonces for challenges.
KryptoKnight has tickets and authenticators, just like Kerberos. Considerable effort has been spent to minimize the number of messages, lengths of messages, and amount of encryption.

The details of the KryptoKnight protocol won’t be discuss here (see [17] for further information). Indeed, the KryptoKnight protocol has many advantages over Kerberos for securing a cluster. The main one is that latency on the cluster network is minimized as the size and the number of tickets are limited. In addition, MAC provides a strong way to check the integrity of the messages exchanged. The initial size of the fingerprint (64 bits) used in the design of KryptoKnight is probably too short for todays use and should be reconsidered to a larger size (for instance, 160 bits as in SHA-1). With this modification, KryptoKnight should be considered as an optimized Kerberos, suitable for our objective in clusters security.

A.4 Standard PKI approach

Except for the SSH approach, architectures like Kerberos or KryptoKnight are based on private key cryptography. Yet, a public key approach can still be considered at the cluster level through Public Key Infrastructure (see [15], chap. 13). General principle of PKI won’t be discussed here. There are in fact

	Point-to-point Channel		Secure architecture for a cluster			Cluster's Grid	
	RSH	SSH	Kerberos	KryptoKnight	PKI	VPN	Globus
Scalability	+++	- -	++	++	+	- -	+
Ressources authentication	-	+	+	+	+	+	+
Communications							
Treatment	+++	-	+	++	- -	++	-
Integrity	-	+	+	++	++	-	++
Non-repudiation	-	-	-	+	+	+	+
Privacy	-	+	+	+	+	+	+

TABLE II

ADVANTAGES AND DRAWBACKS OF SECURITY SOLUTIONS FOR CLUSTERS ANS GRIDS.

various standards for PKI, most of them still in evolution. Examples of PKI that are currently normalized by IETF are PKIX[6] (Public Key Infrastructure X.509), SPKI[7] (Simple Public Key Infrastructure) and DNSSEC (Domain Name System Security).

Yet, we do not think that PKI approach is suitable at the cluster level because it negatively impacts performances. The distribution inherent to PKI infrastructure is not needed for centralized architecture like clusters and leads to lots of unnecessary communications and overheads. Conversely, this approach is rather interesting at the grid level where it's contribution in terms of scalability is essential.

B. Secure architecture for grids

The previous section expounds different way to secure a cluster. We now consider a set of clusters we wanted to bind together. The main difference between a cluster and a grid concerns resilience and scalability. In addition, the grid security policy must be compatible with local policies.

B.1 VPN approach

A first naive idea is to bind the different clusters using the Virtual Private Network technology. In such architecture, a VPN server has to be set at the front-end of each cluster. In practice, this architecture is currently experienced through the French GRID5000 project. Tunnel are there created through the open source project VTun (`http://vtun.sourceforge.net/`) which allows to create tunnels optionally encrypted between the gateway of each clusters. Then each site is an aggregation point for all communication flows; this prohibits the use of simultaneous channels, routes, to be used in parallel. It also implies an explicit connection between every site of the grid which clearly impacts the scale the grid can grow to.

B.2 Globus

Globus [9] is the reference model in secure grid design. It was built from scratch using public cryptography protocols, it assumes the availability of a PKI, and with major goals single sign on and control of access to all grid resources based on capabilities. Additionally to standard security functions including authentication, access control, integrity and non-repudiation, Globus supplies the following functionalities:

- Single Sign On,
- Compliance with local security solutions,
- Protection of credentials,
- Uniform credential/certification infrastructure. The standard used is here X500v3[].

Every resources and clients, be they users or other resources trying to access specific resources, has a certificate that must be signed by a CA in a classical PKI infrastructure way. Single sign on is done using delegated certificates generated on the fly from the users', or resources', certificates. The Globus team is working on standardizing the use of such delegated certificates. In the Euopean DataGrid project [4] modifications were made to the Globus runtime in order to better control the permissions given to a grid user on a local system. This was done using a different mapping than Globus based on ACL (Access Capability List) with a policy defined by each site.

The protocols implemented in Globus and DataGrid won't be further explained here. We will indeed detail in the following paragraph a grid security policy.

B.3 Grid Security Policy

Following [9], the following security policy is proposed:

1. A cluster is a trust domain; the grid is then composed of multiple trust domains.
2. Operations that are confined to a single trust domain are only subject to local security policy.
3. Both global[8] and local[9] entities exist. For each trust domain, there exists a coarse grain mapping from global to local entity.
4. Operations between entities located in different clusters require mutual authentication.
5. Communications between different clusters can be kept private.
6. An authenticated global entity is assumed to be equivalent to being locally authenticated as that local entity.
7. All access control decisions are made locally on the basis of the local entity rights.
8. Programs or processes are allowed to act on behalf of a user and benefit from the user's rights.

[6]http://www.ietf.org/html.charters/pkix-charter.html
[7]http://www.ietf.org/html.charters/spki-charter.html

[8]at the grid level
[9]at the cluster level

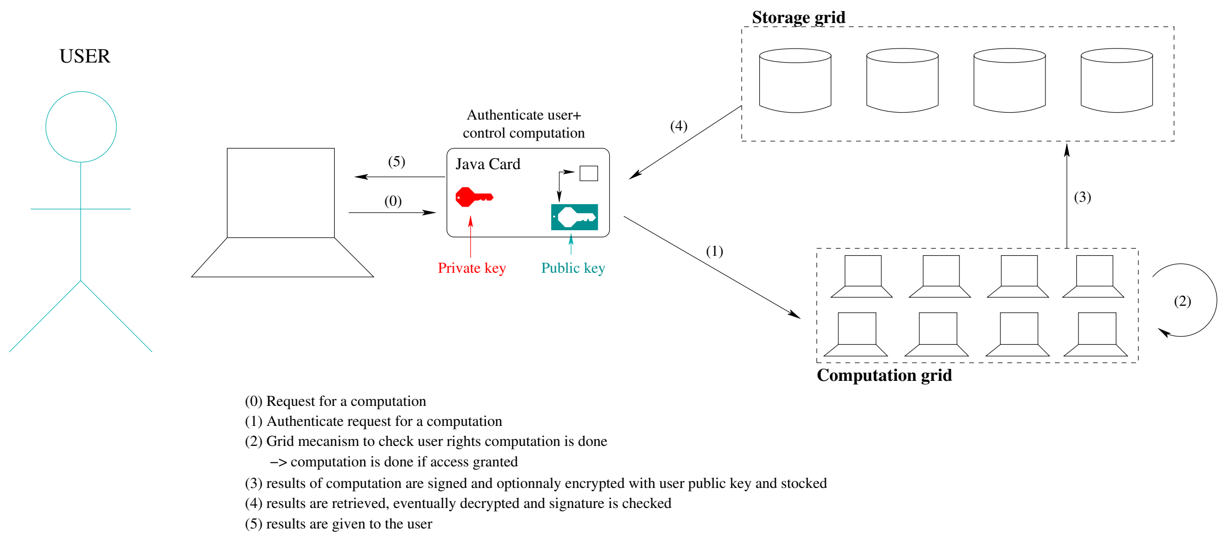
USER
Authenticate user+
control computation
Java Card
Private key
Public key
(0)
(1)
(2)
(3)
(4)
(5)
Storage grid
Computation grid
(0) Request for a computation
(1) Authenticate request for a computation
(2) Grid mecanism to check user rights computation is done
–> computation is done if access granted
(3) results of computation are signed and optionnaly encrypted with user public key and stocked
(4) results are retrieved, eventually decrypted and signature is checked
(5) results are given to the user

Fig. 2. Computation Protocol using smart cards

9. Processes that act on behalf of a user within the same cluster may share a single set of credentials.

C. *Supposed architecture for the grid*

Table II exhibits the advantages and drawbacks of each studied solutions. We can now assume the following architecture for the grid:

- **At the grid level** (inter-cluster relations): Globus, as it is the reference model [10] and the base of the European project DataGrid.
- **At the cluster level** (intra-cluster relation): an effective and adapted architecture, depending on the type of application that is launched on the cluster. For instance, for fine grain application, communications are essential and an architecture that conciliates security and limited latency - such as KryptoKnight - must be privileged. Yet, an experimental evaluation is necessary.

III. Introduction of smart cards in distributed architectures

We now consider a scenario in which a user has access to a computer connected to the Internet. This machine could have been compromised by virus, Trajan horses etc... In addition, the environment set on this computer can be different from the one required for the user computation. We strongly believe that smart cards -and more particularly Java cards- can help to improve security, and more importantly facility on the grid where the computations are done. Firstly because basic properties of smart cards ensure security:

- **at the physical level** by sophisticated impression techniques
- **at the hardware level** by:
  - a unique serial number;
  - the use of PROM memories;
  - a physical armour plat;
  - abnormal condition sensor (temperature ...);
  - the jamming of the information contained in the smart card;
  - cryptographic co-processors
- **at the software level** thanks to access control to data, preservation of data integrity and secure I/O.

Secondly, smart card are easy to use and users have a good perception of smart cards as a private thing such as a key or a password.

The Java Card technology defines a secure platform for smart cards, portable and multi-application that incorporate many of the advantages of the Java language. Security is reinforced thanks to various access control on methods and variables (`public`, `protected` and `private`), a strongly-typed language, the impossibility to construct pointers and a firewall. Only a subset of the Java language is managed. Memory constraints[10] must be taken into account.

We plan to use Java Cards to stock the private and the public keys associated to the owner of the card together with an interface which can launch signed requests on the grid. Therefore, the requests will be authenticated. The computation can be done with respect to the user rights, which are managed on the grid for performance constraints. Results of the computation can then be stocked, signed (and eventually encrypted) with the user public key on a storage grid. The interface on the Smart Card can then be used to retrieve the results, and eventually decrypt them. Figure 2 illustrated this protocol.

Experiments on Java Cards are still in progress. Yet, even if this is an early technology, the fact that it can make the life of the grid user easier reinforce our belief in its acceptance as a main functionality in future grid environments. We also plan to include this technology in the architecture presented in the following section. That architecture extends the one expounded in §II to deal with tasks forgeries.

IV. Result Checking in Peer-To-Peer Computing

The architecture proposed in §II solved the following issues:

- Single Sign On.

[10] 1Ko RAM, 16 Ko EEPROM and 24 Ko ROM

- Authentication of users and nodes.
- Privacy, integrity and non-repudiation of the communications

Even on such a secured environment, the outputs of the program that is executed on a remote resource can be modified with no control of the client application. In all this paper, a task is said *forged* (or *faked*) when its output results are different than the results it would have delivered if executed on an equivalent resource but under the full control of the client. This may occur when the remote resource is the victim of a Trojan horse or if the client software is modified on the remote resource, as experienced with SETI@Home [16].

Result checking is then a way to detect and eventually correct faked tasks of the program that is executed on the cluster. In literature, software certification of results is achieved in two ways:

1. The **”simple checkers”** approach [6] consists in verifying computed results thanks to a post-condition easy to compute. This approach is simple and elegant. Though, it is often impossible to automatically extract such post-condition on any program. Furthermore, if a computation is performed on numerous peers, the detection of a faked final result does not supply any information on the peer(s) responsible for the forgery.
2. To tackle this problem, the **”duplication”** approach [19] is based on several executions of each task on various resources. Duplicating all jobs would generate an important additional cost. To limit it, C. Germain and N. Playez [12] propose a probabilistic certification based on a sequential test of Wald [21], only a few randomly chosen tasks are checked. This approach is limited for the case of independent tasks.

A. Generic Partial Post-Condition

In [13], S. Jafar & al. extend this approach for the case of tasks with dependencies, thanks to Data-Flow Analysis. In that framework, the application is represented by a bipartite direct acyclic graph G: the first class of vertices is associated to the tasks whereas the second one represents the parameters of the tasks (either inputs or outputs according to the direction of the edge).

A leaf parameter in G is called a *terminal output*. Associated to a set of terminal outputs $\mathcal{S}$, the *terminal subgraph* is the subgraph $G_{\mathcal{S}}$ restricted to the ancestors of the vertices in $\mathcal{S}$. Figure 3 illustrates those notions. Note that $G_{\mathcal{S}}$ can be computed from G in linear time $O(|G|)$. A complete execution of the program supplies then a graph $G_{\mathcal{S}}$ in which all the parameter values are explicit, as in fig.3. This graph is called the **execution track** of the program.

Assuming the existence of at least one trusted machine (also called *oracle*), we compute on it a partial data flow graph where all the parameters values are symbolic, except the input parameters of the program ($\{e_1, ..., e_4\}$ in fig.3) This partial graph is a summary of the execution track called the **certification track**. It only describes the tasks to be executed and their dependencies. It has been generated on reliable resources (oracles) and verify the following properties:

Proposition 1. • *the certification track is a summary of the execution track;*
- *a partial execution is sufficient to generate it;*
- *any correct execution of the program (with the same inputs) on a remote unsecured worker supplies an execution track which summary has to correspond to the certification track.*

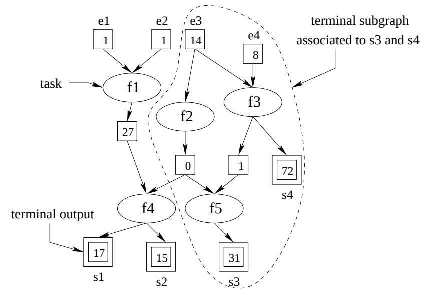

Fig. 3. Instance of a data-flow graph associated to the execution of five tasks $\{f_1, ..., f_5\}$. The input parameters of the program are $\{e_1, ..., e_4\}$ whereas the outputs (i.e the results of the computation) are $\{s_1, ..., s_4\}$.

Therefore, a partial post-condition that can be applied to any program is defined. Even if it does not allow to certify the reliability of the computation, it makes it possible to control the general structure of the executed DAG.

Besides, once this post-condition is verified, the execution track can be used to certify the set of terminal outputs $\mathcal{S}$ to detect (and eventually correct) attacks which do not change the structure of the DAG. Many strategies are possible and are still described in [13]. The following section proposed an architecture that implements this partial post-condition and allows all these strategies to be done.

B. Distributed software architecture for certification

In this section, the source code of a program to certify is considered. The previous notions are integrated in a software architecture to provide the certification of this program.

Figure 8 gives an overview of the proposed architecture. This infrastructure can be divided in four modules which are associated to the four steps of the certification :

1. `Compilation_Module` : this module receives the initial source code of the program to certify (`prog.c` in figure 4) and provides three codes that will be used by the other modules :

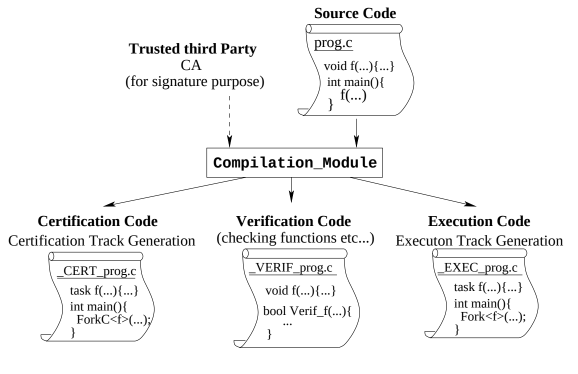

Fig. 4. The `Compilation_Module` generates three codes that are used by the other modules

• *Certification Code* : the execution of this code provides a partial data-flow graph - i.e. the certification track of the initial programme. This graph requires only a partial execution of the source code to be generated.

• *Execution Code* : similarly, the execution of this code provides the execution track of the program which is a representation of the data-flow graph related to the execution of the source code.

• *Verification Code* : this code contains all the checking functions. These functions achieve the atomic re-execution of a given task and a comparison of the results of this re-execution to the one extracted from the execution track.

The generation of those codes can be automated thanks to Macro-Data-Flow API such as Athapascan [11] which we use in our current implementation.

This module corresponds to an initialization step and has to be executed in a secure environment. The different codes it provides have to be signed by a trusted third party for further authentication.

2. `PCG_Module` for "Program Certificate Generation Module" : this module can be optionally executed to provide a certificate of the program to certify. This certificate can then be exported for the case of a remote certification of the program on a remote secure oracle.

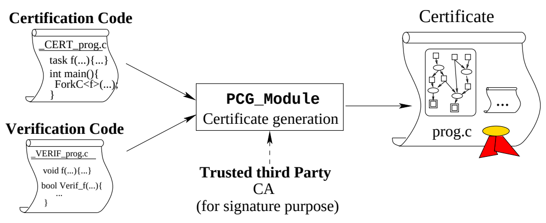

Fig. 5. The `PCG_Module` (Program Certificate Generation Module) creates a certificate for the program to certify for the case of a remote certification

As mentioned in figure 5, this certificate is constituted by a representation of the certification track and the verification code. This certificate is also signed by a trusted third party for authentication purpose. This step can be skipped if the certification module described later uses the same trusted resources than the compilation module. Otherwise, the certificate can be exported.

3. `Execution_Module` : this module submits the Execution Code to the clusters grid it is linked to. The way the tasks are scheduled is not detailed here. Examples of batch scheduler can be found in [7].

As described in figure 6, the computation provides the execution track that will be submitted to the Certification Module.

4. `Certification_Module` : this module is responsible for the certification itself. Its behavior is detailed in figure 7.

The main inputs for this module are :

• the certification track and the Verification Code (eventually extracted from the program certificate);

• the execution track provided by the execution module which represents the program to certify;

• the certification algorithm to use after the checking of the

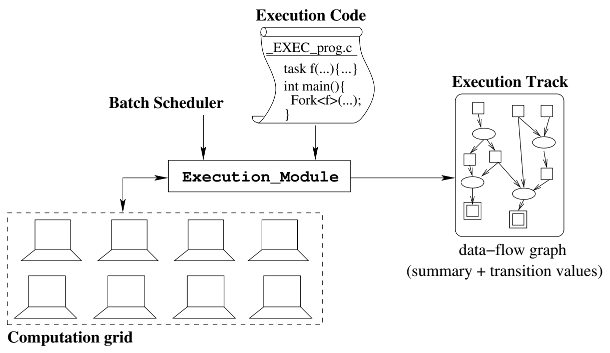

Fig. 6. The `Execution_Module` manages the execution on a computational grid and provides the execution track to certify.

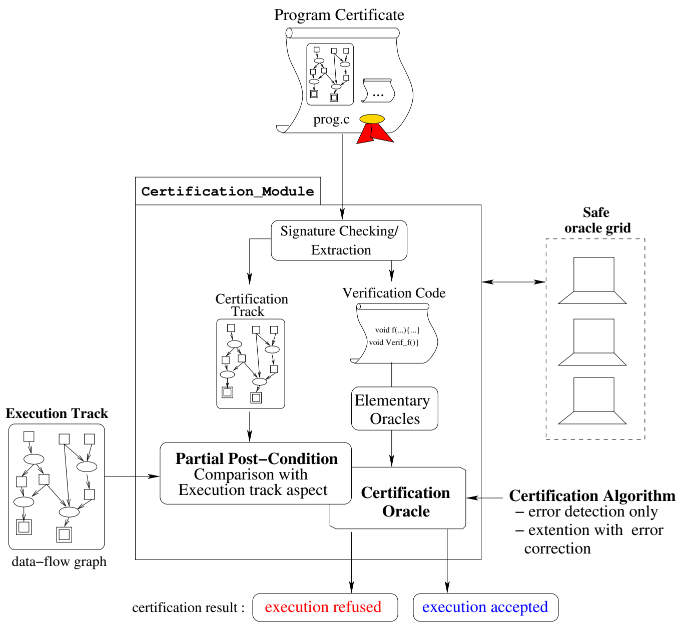

Fig. 7. The `Certification_Module` certifies the execution track provided by the Execution Module. It decides whether the computation is accepted or rejected due to forgery detection.

partial post-condition defined in §IV-A. This algorithm is used to check the set of terminal outputs of the program to certify. Possible algorithms are described in [13].

This module is connected with a safe grid where oracles computations can take place. It decides whether the terminal outputs to certify are correct or not. The partial post-condition checks the aspect of the tracks whereas the certification algorithm checks a set of terminal outputs to certify. Both are independent and can be done in parallel.

Figure 8 proposed a global view which exhibits the relations between the modules. In the next section we analyze robustness of this certification with respect to attacks.

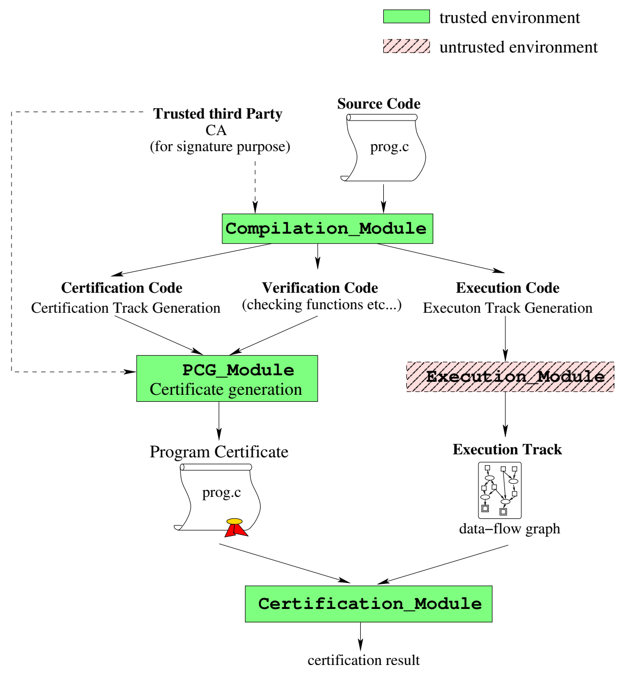

Fig. 8. Global view of the proposed architecture

C. *Robustness to attacks and resilience*

Historically, the first infrastructure which highlighted the certification issue was the SETI@Home project [3] in 1999. Whereas the project succeeded beyond the wildest dreams of its creators, people who believed the SETI@Home client software too slow decided to provide a patch to makes the client faster [16]. The previous architecture would have managed to detect the patched clients thanks to the partial post-condition checking.

By using a trusted third party which signs the certificates and the codes generated by the Compilation Module, this architecture provides solutions for authentication and integrity checking. Confidentiality can also be set thanks to SSL protocols for example. By the way, usurpation threats and snooping attacks can be avoided.

Yet, DoS attacks are still dangerous on this architecture (like many others), more particularly if the safe grid used for the oracles can be targeted.

It introduces the issue of the resilience in the nodes availability. Classic solutions implements periodic challenges (or with an adaptive step). Typically, an authentication challenge with public key as the one used in SSL is considered. Such challenge allows to guarantee not only the presence of a resource but also its authentication.

Nevertheless, an alternative could be to assimilate the challenge to a particular computation which is not different than a real computation for the worker point of view. Such tasks won't be duplicated during the certification. Under these assumptions, challenges checking allows to estimate the confidence to place in the distant resource.

V. Conclusion

In this paper we have over-viewed security infrastructures proposed for clusters and grids. Focusing on their potential use in the case of a large scale global computing platform, we provide a classification according to the following criteria: scalability which is the critical point to address; level of authentication; treatment, integrity, non-repudiation and privacy.

We have exhibited three critical security points to address for a global computing platform that consists in a dynamic grid of clusters:
- the security of the whole global platform has to be consistently built upon the various local policies on each individual platform;
- authentication of a user should be possible from any machine, even a corrupted one;
- the results delivered by any user application have to be certified, even if some resources or some application processes may be corrupted.

Based on previous results on result checking and on the availability of smart cards to enable authentication from any peer, we have proposed a global security infrastructure that tackles those problems. This architecture assumes the use of identified trusted machines (called oracles) dedicated authentication and certification of results.

Certification of the execution is based on the representation of the application execution by a data-flow graph that describes both computations to perform and their dependencies. Such a representation is inspired from fault-tolerance systems on heterogeneous architectures [13]. We detail the use of this representation as a certification track; then it can be seen as the hashing of the execution. Also, this track provides a generic postcondition that can be checked based on partial replication of computation tasks.

In the framework of a cooperation between Université du Luxembourg and Institut National Polytechnique de Grenoble, a prototype of this global security infrastructure is currently implemented on a grid built form clusters in France (Grid'5000 national project) and Luxembourg. Target application involves medical computations (based on medical images comparison) studied within the Région Rhône-Alpes RAGTIME project [2]; this application asks also legislation issues not addressed in this paper.

References

[1] "BlueGene," http://www.llnl.gov/asci/platforms/bluegenel/.
[2] "The RAGTIME Project," http://dionysos.univ-lyon2.fr/ miguet/ragtime/.
[3] "The SETI@Home project," 1999, http://setiathome.ssl.berkeley.edu/.
[4] "The European DataGrid project," 2000, http://web.datagrid.cnr.it/.
[5] S. M. Bellovin and M. Merritt, "Limitations of the kerberos authentication system," *SIGCOMM Comput. Commun. Rev.*, vol. 20, no. 5, pp. 119–132, 1990.
[6] M. Blum and H. Wasserman, "Software Reliability via Run-Time Result-Checking," *Journal of the ACM*, vol. 44, no. 6, pp. 826–849, Novembre 1997.
[7] N. Capit, G. H. Georges Da Costa, C. Martin, G. Mounié, P. Neyron, and O. Richard, "Expérience autour d'une nouvelle approche de conception d'un gestionnaire de travaux pour grappe," in *CFSE'3*, La Colle sur Loup, France, Oct. 2003, pp. 602–613.
[8] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "Grid services for distributed system integration," *Computer*, vol. 35, no. 6, 2002, http://www.globus.org.
[9] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids," in *Fifth ACM Conference on Computer*

and Communications Security Conference, San Francisco, California, 3–5 Novembre 1998, pp. 83–92.

[10] ——, "Security for grid services," in *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, I. Press, Ed., Seattle, Washington, 22–24 Juin 2003.

[11] F. Galilée, J.-L. Roch, G. Cavalheiro, and M. Doreille, "Athapascan-1: On-line Building Data Flow Graph in a Parallel Language," in *International Conference on Parallel Architectures and Compilation Techniques, PACT'98*, IEEE, Ed., Paris, France, Octobre 1998, pp. 88–95.

[12] C. Germain and N. Playez, "Result checking in global computing systems," in *Proceedings of the 17th Annual ACM International Conference on Supercomputing (ICS 03)*, ACM, Ed., San Francisco, California, 23–26 Juin 2003.

[13] S. Jafar, S. Varrette, and J.-L. Roch, "Using data-flow analysis for resilence and result checking in peer to peer computations," in *Proceedings of the 15th International Workshop on Database and Expert Systems Applications (DEXA 2004) (To appears)*, Zaragoza, Spain, september 2004.

[14] C. Martin and O. Richard, "Parallel launcher for cluster of pc," in *ParCo'2001*. Napoli, Italy, 2001, pp. 473–480.

[15] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, C. Press, Ed. CRC Press, Inc, 1997.

[16] D. Molnar, "The SETI@Home Problem," November 2000.

[17] R. Molva, G. Tsudik, E. Van Herreweghen, and S. Zatti, "Kryptoknight authentication and key distribution system," in *Proceedings of European Symposium on Research in Computer Security (ESORICS)*, Toulouse, France, november 1992, pp. 155–174.

[18] C. Neuman, T. Yu, S. Hartman, and K. Raeburn, "The kerberos network authentication service (v5) ietf," USC-ISI/MIT, Tech. Rep., February 2004.

[19] L. F. G. Sarmenta, " Sabotage-Tolerance Mechanisms for Volunteer Computing Systems," in *ACM/IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01)*, Brisbane, Australia, Mai 2001.

[20] M. Sirbu and J.-I. Chuang, "Distributed authentication in kerberos using public key cryptography," in *Symposium on Network and Distributed System Security*, San Diego, California, 1997, pp. 134–143.

[21] A. Wald, *Sequential Analysis*. Wiley Pub. in Math. Stat., 1966.