Trust in Distributed Networks

Siegfried Podesser

Trust in Distributed Networks

Master Thesis

at

Graz University of Technology

submitted by

Siegfried Podesser

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology
A-8010 Graz, Austria

12th January 2010

© Copyright 2010 by Siegfried Podesser

Advisor: Univ.-Prof. M.Sc. Ph.D. Bloem, Roderick Paul
Co-Advisor: Dipl. Ing. Tögl, Ronald

Abstract

The term grid computing originated in the early 1990s in Ian Foster's and Carl Kesselman's seminal work as a metaphor for making computing power as easy to access as an electric power grid. In 2007 the term cloud computing came into popularity, which is conceptually similar to the canonical Foster definition of grid computing (in terms of computing resources being consumed as electricity is from the power grid). Grid/Cloud computing is a very strong growing technology. Massive computation power directly from the internet-socket is a very popular advertising message. It means that everyone who has access to the internet can access grid/cloud computing services and use their computation power for calculations. Currently there are a lot of open source and commercial solutions.

However, a number of main security concerns exist in current grid/cloud computing solutions, which cannot easily be solved with conventional approaches:

1. Participants may deliberately report back wrong answers.
2. Participants may not enforce data integrity or not compute accurately.
3. Data protection requirements of sensitive data cannot be enforced on remote systems.
4. Theft of intellectual property on the distributed code by the participants is possible.
5. Theft of confidential data by the participants is possible.

The main problem with all current grid/cloud computing solutions seems to be the complete or at least in parts missing trust-service. For example Permis, a Java based authorization and credential validation service makes it possible to add an authorization and/or a credential based validation service for grid networks. Nevertheless this service still misses a 'trust' part which let this software proof its trustworthiness. Furthermore a security feature like securing the computed data from stealing is nearly missing in most solutions.
As recently proposed, the already widely distributed TPM (Trusted Platform Module) could be used to add 'trust' into grid/cloud computing. To do so is the main goal of this thesis.

The first part of this thesis will give an overview about the different types of grid/cloud computing software that already exists. There will be a short summary how the already existing grid/cloud computing solutions are attempting 'trust' into their grids. We will focus on open source based ones, because of the possibility to take a closer look inside the code. The main theme in the theoretical part is the study of existing middleware and the design of a proper PKI for grid/cloud computing systems and for Java based applications.

In the practical part of the thesis we demonstrate and discuss a solution of how to combine GridGain and Permis and how to add further 'trust' into such a new combined framework.
This example solution is adaptive for many grid/cloud computing applications, due to the widespread usage of Permis. The reason for choosing GridGain as grid/cloud computing framework is that it is based on Java, well documented and well programmed. Furthermore in the practical part of this work JSR321 is used to add more trust into our grid/cloud computing solution (with help of the TPM).

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Ort	Datum	Unterschrift

Contents

Chapter 1

Introduction

Grid computing and cloud computing are operating in a field where problems are no longer solvable by one single computer. The amount of data is simply overwhelming. The CERN [4] for example can produce terabytes of data in a few milliseconds. Where can it be stored without loosing any information in that short time? SETI@Home [7] has an other kind of problem to offer. For more than 10 years they have been listening for narrow-bandwidth radio signals from space. The amount of data which is produced every second is simply gigantic and it takes hours to analyze it and to search for extraterrestrial signals in it. A combination of both problems is faced by Google [11] with its gigantic and very fast search engine. With just one single search request more than thousand computers get busy to search the enormous data of the whole web. Grid and cloud computing are offering the solutions for these problems.

Currently all the public grids and clouds have enormous security concerns like the ENISA report [15] has shown. Private clouds are at risk as well, even if they are better secured and are not accessible through the Internet. We believe that Trusted Computing can improve grid and cloud security.

Therefore our solution will at first analyze four well-known grid and cloud computing solutions against the following five main security concerns which are directly taken from [49]:

1. Participants may deliberately report back wrong answers.
2. Participants may not enforce data integrity or not compute accurately.
3. Data protection requirements of sensitive data cannot be enforced on remote systems.
4. Theft of intellectual property on the distributed code by the participants is possible.
5. Theft of confidential data by the participants is possible.

After that analysis we will construct our own grid/cloud computing framework to overcome most of these main security concerns. Therefore we will select at first our software components, then design our public key infrastructure and at last we will make our Proof-of-Concept which will show that such a grid/cloud computing solution can be really built in software.

This thesis will guide the reader at first in chapter two through the needed theoretical background in the fields of grid and cloud computing, the very basics of cryptography, what public key infrastructures are representing, what access control in computer networks means and what trusted computing is all about.

Chapter three then centers about the two very popular grid computing solutions BOINC and folding@HOME. Both are quite similar in offering a public software which allows anyone to join their computational network and to share their private unused computation power with the network. This kind of software is quite easy to use for everyone. Simply start it, choose your computational project or let

it be chosen automatically and then share your free computation time. Different to those programs are the Globus Toolkit and GridGain which are offering a complete grid and cloud computing framework. Out of the box they are not runnable at all. This means that it needs at least one developer who forms the right parts of the framework together and puts the program logic into the new to be created application. Conversely these frameworks are much more powerful than the first two programs because of their all-round usability. In the end of chapter three a general overview of risks for computational networks is given and how the before described programs are able to deal with them.

To deal with the shown risks an integrated solution for trustworthy grid and cloud computing is developed through chapter four. Therefore this chapter presents at first an overview about the idea of a new and more trustworthy framework. This idea is then followed by a very fine grained discussion of the three main parts of our new framework: GridGain, Permis and the JSR321. GridGain itself offers a very powerful and modular Java based grid/cloud computing framework, but it has simply no trust mechanisms at all. Therefore Permis, a modular, very powerful and highly complex authorization framework together with JSR321, which implements trusted computing functionality for Java, are added to form our trusted grid framework.

The simple combination of those three programs will not bring that much more trust into the framework without an appropriate Public Key Infrastructure (PKI). Exactly that PKI is theoretically developed through chapter five by giving at first a conceptual overview of what will be done, followed by a component description, which shows accurately all elements of this PKI with an appropriate description. Then follows a PKI concept description and at last a risk analysis. The chapter concludes how the new framework together with this PKI is able to handle that risks.

Chapter six presents a practical proof of this conceptual PKI and framework design. It shows quite accurately how the registration process can be implemented. Further we present how GridGain itself must be modified to work with Permis and the JSR321 together and how the internal work can look to achieve the before conceptually described and shown features practically. The end of this chapter forms the practical overview how the server and the client for GridGain can be built.

During Chapter seven, which is called “Experiments”, an overview about the workflow between the server and the clients is given. Furthermore the annotation feature for the workunits and the setup for the experiments is shown. The last Section in this Chapter is dedicated to the performance issue.

The last chapter gives an overall view about the problems which have been found in the reviewed grid/cloud computing applications. In relation to these problems a short overview about our Grid/Cloud Computing framework in theory and practice is given. Further a short summary about the directly measurable differences between the original GridGain and our new Grid/Cloud Computing framework is presented.

The appendix at the end of this thesis holds a lot of different hints and a full how-to guide, which describes very accurately how the different components must be installed and configured to get them running.

Chapter 2

Theoretical Background

This chapter introduces all used technologies in this thesis and gives a short overview about them. The understanding of these technologies is essential to have the needed background to understand each chapter and section of this thesis.

2.1 Grid Computing

The term grid computing originated in the early 1990's in Ian Foster's and Carl Kesselman's seminal work as a metaphor for making computing power as easy to access as an electric power grid [22]. The idea behind Grid Computing is to use more than one computer to solve big computational problems or to store large amounts of data in short time. Computational problems which are simply unsolvable on a single computer become solvable by using a big network of such simple computers.

The best known example of grid computing is Google [11]. For its search engine Google uses more than 200.000 custom-built commodity servers [11][39]. The idea behind using common computers for forming its monstrous clusters is the availability of cheap hardware, the better energy efficiency than using always the latest technology and the simple interchangeability of such a hardware. The basic principle of Google's fast responses for any queries is the giant parallel handling of any query to its cluster. A simple query is divided into a multitude of parallel calls to different single computers in this cluster. Moreover Google uses its framework not only as a computational grid but further as a data grid. All the data that have been indexed by Google are spread over all its common computers. That is the reason why Google can search, compute and answer so fast to any given query.

2.1.1 Computational Grids

"Computational Grids account for a lion share of Grid Computing usage now and will certainly retain this lead in the near future to the least." [9]
The idea behind computational grids is to split large and long time running tasks into multiple sub tasks. Each sub task can then be executed on a different computer. If all sub tasks have been solved then the original task result can be achieved by combining all the sub task solutions together.

Further it allows a very flexible way of reducing the risk of system failure of one computer. This failure simply does not influence normally the other computers in the grid.

2.1.2 Data Grids

The CERN (the European Organization for Nuclear Research) uses one of the biggest data grids [23] in the world. The data grid must be capable to save (from an experiment that happens in a few milliseconds)

hundreds of TeraBytes to PetaBytes in a very short time frame. The idea behind data grids is not only to store that kind of data rapidly but to access them as well very quickly. Data grids are therefore very similar to computational grids, which divide large tasks into smaller sub tasks. They divide large amounts of data into smaller pieces, store them on multiple computers and allow anyone who is authorized to access them as one single piece.

2.2 Cloud Computing

Cloud computing is the next logical development step of grid computing for the common user. It abstracts all technological intricacies and shows only an abstracted interface to the end user. Its purpose is to allow them the usage of grid computing without any knowledge about how to use this technology.

An example for such a cloud computing environment could be the usage of a web office package. The end user uses then the computational power of an abstracted and hidden grid to write, to edit and to save his documents and nothing of that is visible to him. Another example can be Google again. If a user makes a query then he triggers a whole set of actions that the computational grid of Google has to handle. But nothing of that is visible to the end user. Only the answer of the query shows up.

A second description for Cloud Computing [38] could be the following structure-centered view, which looks at Cloud Computing (see figure 2.1) as on a three-step pyramid with the following basic elements: IaaS, PaaS and Saas.

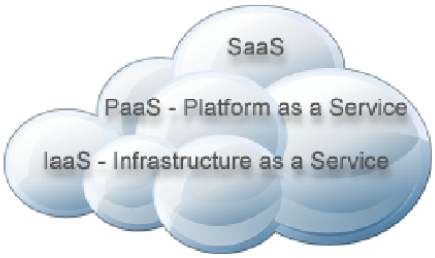


Figure 2.1: SaaS, PaaS and Iaas [38]

2.2.1 IaaS

The ground stage of the Cloud Computing pyramid is represented by the Infrastructure as a Service (IaaS) layer. This stage is responsible for all kind of infrastructure, which is needed for the cloud, like the backup or archiving systems. The main task is to provide a wide range of virtual execution environments like virtual machines with operating systems and virtual desktops or simply computational power. The main advantage of this layer is the scalability.

2.2.2 PaaS

Based on the IaaS layer the Platform as a Service (PaaS) layer is fitted above. This layer delivers the computing platform which includes all kinds of software components like databases, application servers or other software systems. The scope of this layer is to offer all needed functions for the target service which is running on the top of this layer.

2.2.3 SaaS

The top of the Grid Computing pyramid forms the Software as a Service instance. One software is developed, operated and maintained by one company for example. An arbitrarily end user only needs a key or an account to access that software service. This is the layer which is the most known one by the end users. The example before with the web office package is further too one for this layer.

2.3 Cryptography

Cryptography is the science of ciphering information and therefore it uses the power of mathematics to encrypt and decrypt data. Encryption means the transformation of plaintext (or human readable text) into ciphertext. A typical encryption from plaintext into ciphertext and back to plaintext can be seen in figure 2.2.

Such a cryptographic technique can be called strong or weak. A weak cryptographic technique is one that can be easily broken by cryptanalysis. Cryptanalysis deals with the breaking of encrypted data. Therefore it uses a combination of analytical reasoning, pattern finding, patience, determination, application of mathematical tools and luck. The umbrella term for cryptanalysis and cryptography is called cryptology. A strong cryptographic technique is defined as an encryption that simply isn't decryptable within a reasonable time. Further the used algorithms are public and proven. That means that if any available today's computational power is used for breaking the encryption and all the known methods from the Cryptanalysis are used, the Cryptography would still be unbreakable in a reasonable span of time.

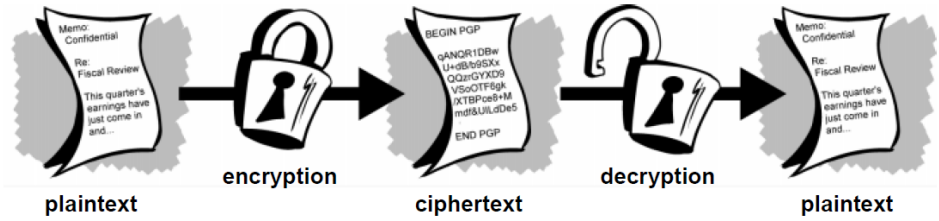


Figure 2.2: Cryptography - From plaintext to ciphertext and back. [33]

Today's Cryptography knows two kinds of keys for the encryption and decryption process: symmetric cryptography and asymmetric cryptography.

2.3.1 Symmetric Cryptography

The symmetric cryptography uses the same key for the encryption and the decryption (as shown in figure 2.3). This key is called secret or symmetric key. The DES or Data Encryption Standard [48] for example uses a symmetric key. The main disadvantage of symmetric keys is that the same key is used for decryption and encryption. Therefore always the problem with the secure key distribution must be solved. State of the art in symmetric cryptography is at the moment the AES or Advances Encryption Standard [17]. The direct opposite to the symmetric cryptography is therefore the asymmetric cryptography (see next subsection). For large amounts of data is the symmetric cryptography still the only usable one. The reason is simply the smaller amount of CPU utilization which is needed for the encryption and decryption process. Therefore it is usual to use symmetric cryptography for large amounts of data and asymmetric cryptography for securing small amounts of data like for example the symmetric keys.

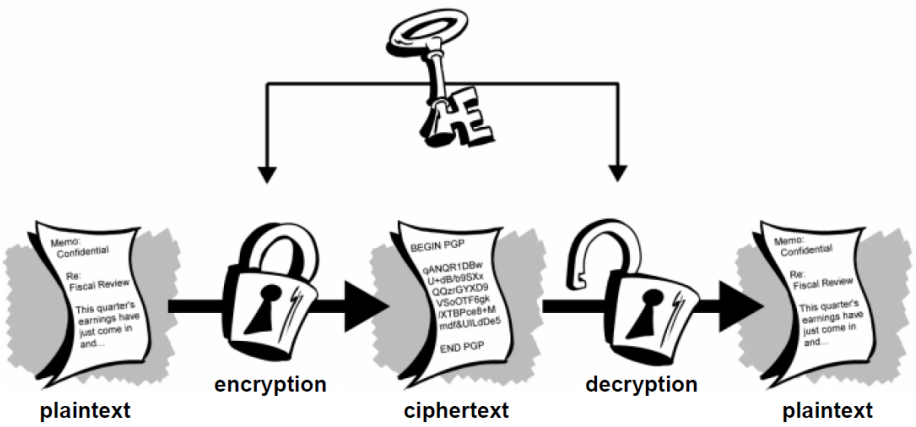


Figure 2.3: Symmetric Cryptography [33]

2.3.2 Asymmetric Cryptography

Asymmetric cryptography uses two kind of keys, the private one and the public one (as shown in figure 2.4). Those two keys are directly dependent on each other. While the public key is usually given to anyone who would like to send encrypted data to the owner of that key, is the private key only in property of the owner itself. The private key never leaves its owner. Therefore the owner of the private key is the one and only who can decrypt data which has been encrypted by his public key. The public key is completely mathematically independent from the private key. This asymmetric cryptography is also known as public key cryptography. State of the art in asymmetric cryptography are currently the RSA [25] and the ECC or elliptic curve cryptography [25].

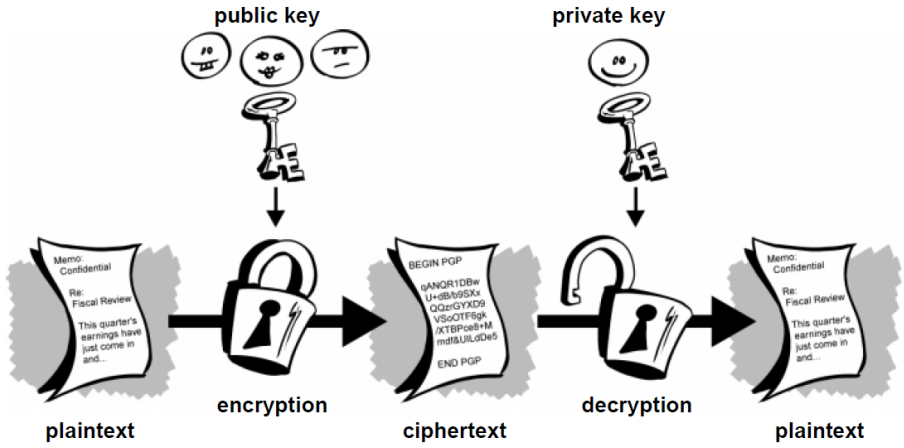


Figure 2.4: Public Key Cryptography [33]

2.3.3 Cryptographic Hash Functions

This section is based on [32]. In cryptography, hash functions are used to ensure integrity, authenticity and non-repudiation. Integrity is used to ensure that the data can't be modified without realizing it. Authenticity is used to guarantee that the given data is explicitly from a certain person or entity and non-repudiation guarantees that the producer of the data can not deny the data at a later date. This is done by a deterministic procedure which transforms a given message of any size to a string with a fixed size. This fixed string can be seen as the fingerprint of the given message. A good cryptographic hash

function should create for any different message a different hash string, which is obviously not possible. This ensures that any change of a given and hashed message can be detected and therefore its integrity, authenticity and non-repudiation can be ensured.

SHA1 represents such a Cryptographic Hash Function. It transforms a given message into a 160-bit string. Recently more and more scientific research in the area of collision attacks against the SHA1 has been done[51]. It has been shown that the edge of successful and complete collision attacks come closer and closer. That is the reason why SHA2 should be used instead of SHA1. SHA2 transforms any given message into a 256-bit or 512-bit string and until now no collisions have been found.

2.4 Public Key Infrastructure

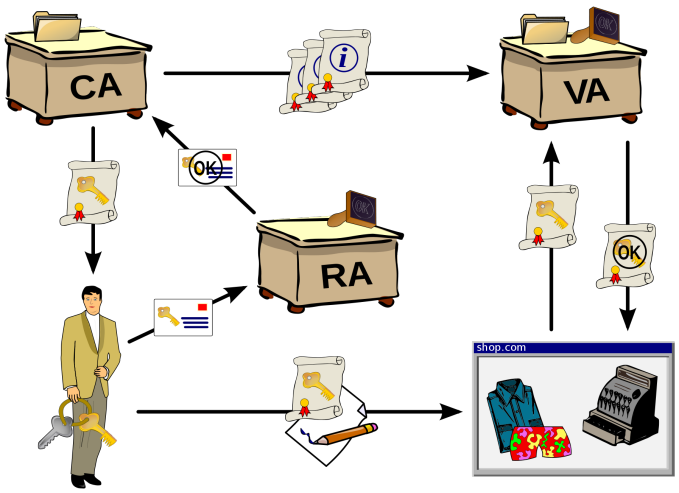


Figure 2.5: Public Key Infrastructure [1]

2.4.1 Overview

A Public Key Infrastructure[50] consists of a framework for the creation, validation and revocation of keys. Therefore it is represented by a set of hardware and software to achieve the following goals in an insecure environment:

1. Confidentiality: Sensitive data must be protected in a way that only authorized persons can get access to them.
2. Integrity: To prevent data of getting corrupted or modified and to achieve that transactions can not be altered integrity is needed.
3. Authentication: Authentication in relation to a PKI means that every entity must be verified through its given public key certificates and its digital signatures to verify that the requested claim is correct. The main goal is to allow unknown entities to authenticate to each other by using public key certificates.
4. Non Repudiation: It must be ensured that it is always explicitly clear who has signed the data without any doubt.

These four goals can be achieved with the following approaches:

1. To create confidentiality cryptographic encryption mechanisms can be used. The encryption itself can be asymmetric (public/private key) or symmetric (with a secret key). Asymmetric cryptography needs much more computational resources than symmetric cryptography. That is the reason why normally only a small amount of data is encrypted asymmetric, like secret keys used for symmetric encryption systems.
2. Integrity can be achieved by using public key certificates and digital signatures to envelop the data. Within a PKI the integrity can be provided by the use of a asymmetric or symmetric cryptography. Generally a SHA-1 or MD5 hashing algorithm is used in conjunction to guaranty the integrity.

3. The authentication process itself in a PKI environment is originated on a mathematically relationship between the public and the private keys. Messages that have been signed by one entity can be verified by any relying entity. Confidentiality is created through this process because of the reason that only the owner of the private key can create appropriate messages and only he has access to the private key.

4. Non Repudiation is a by-product of using asymmetric cryptography. It ensures that data can't be renounced or a transaction be denied. Everyone who has access to the public key can verify the digitally signed data.

In other words a Public Key Infrastructure represents a system for binding a public key to a specific and distinct user identity. This binding is done by the registration authority (RA) which validates for every unique user identity its public key, the user information, the validity information and further attributes. After the successful registration at the RA it delivers the user a binding package to the Certificate Authority which creates the public key certificate for him, signs it with the CA certificate and delivers it then to the user.

2.4.2 Properties of a Public Key Infrastructure

As shown in Figure 2.5 a Public Key Infrastructure consists at least of the following elements:

- Digital Certificates: Digital Certificates represent digitally signed data which is needed to verify the authenticity of an object. With the help of the signing process a CA guarantees the authenticity of the presented and signed data and therefore it can be used to securely authenticate a person. Further a digital certificate guarantees the integrity, the authenticity and the non repudiation of a signed message. Because after signing the message, the signer can not deny it and further the message itself can not be changed without destroying the signature of the signer.

- Certificate Authority (CA): The Certificate Authority represents the organization that manages, distributes, stores and revokes the digital certificates. Further it is the organization which delivers trust into the signed certificates. This is done by an excellent reputation of this organization and by signing only data which comes from the Registration Authority, which is in charge of establishing the identity.

- Registration Authority (RA): The Registration Authority manages the registration of new persons, computers or secondary registration authorities. Therefore the given data from these groups is verified against correctness for the desired certificate. If successful, then the Certificate Authority signs it.

- Certificate Revocation List (CRL): This list contains all certificates that have been revoked before the end of there validity. Possible reasons for the revocation could be for example a compromised key or wrong registration data.

- Directory Service: The Directory Service contains all issued certificates. Normally it is implemented as a lightweight directory access protocol (LDAP) or sometimes as a X.500 server service.

- Validation Authority (VA): The Validation Authority offers a service for the validation of issued certificates in real time. This check proofs the integrity and the authenticity.

2.4.3 Alternative PKI Approaches

Hierarchical

The most typical implementation of a Public Key Infrastructure is the hierarchical one [14]. This model is build like a tree with at least one root Certificate Authority (CA) and arbitrarily many sub CA's. All the rules and conventions that are predetermined by the root CA must be adopted by its sub CA's. A sub CA can only add further restrictions, rules or conventions but not reduce them. The benefit of this system is that any sub CA inherits the trust of the predecessor CA until the root CA is reached.

Distributed

A complete contradiction to the Hierarchical PKI model is the Distributed Web of Trust [14]. In the Distributed Web of Trust no CA or a trusted third party can be found. This system, which has been introduced through Pretty Good Privacy (PGP) [37], uses this kind of a PKI for its trust model in email environments. This model does not scale very well because every entity only defines for itself who is trustworthy and who not. If an entity thinks that certificate A is really from the person A then he signs this certificate with his own one. The more signs a certificate has the more reliable it seems that this certificate is really from this person. This system has a lot of weaknesses and is not really usable as a commercial PKI. Therefore this PKI cannot be used efficiently to build an automated system for trust decisions based on a TPM system.

Direct

The direct trust model [44] is very similar to the Distributed Web of Trust model. The trust between two peers is based on a symmetric key. A trusted third party is not needed in this system, because any peer has to handle its keys and secrets by his one. For a commercial or at least a bigger PKI this system is nearly unusable. A global revocation of one or more keys is further not realizable.

Cross Certification

In a world wide environment a single CA may be insufficient to be able to verify the validity of all existing certificates. Therefore the Cross Certification method [19] offers entities the possibility to use more then one CA together. The selection of the CA's is based on the particular needs of the entity. This system works like this: An entity creates his own digital certificate and every needed CA performs its various diligence tests on it. These tests would check for example if the given certificate fulfills the published Certificate Policy of the current CA. If the tests are passed successfully then the CA cross-certifies the certificate. If now an entity uses this cross-certificate to authenticate itself to a sub CA of one of the used CA's then the CA will acknowledge the signing from its root CA and therefore it can verify if the signing is valid. Further any CA can handle their own revocation list and therefore revocation itself is easily realizable.

2.5 Access Control in Computer Networks

This section is based on [20]. Access control means that only certain authenticated and authorized entities should be allowed to access a protected and non-public resource. Therefore this resource must be locked and a guard must be installed that takes control over this resource. Only users that can authenticate and show permissions to the guard will be able to access the resource.

There are a lot of different access control systems but they all have one thing in common, it's the way how they make access control decisions. The access control decisions on such systems are always based at least on one of the following three facts:

1. ownership: Something the user has (e.g., RFID chip, identity card, security or software token, ...)
2. knowledge: Something the user knows (e.g., password, pass phrase, social security number, ...)
3. biometric: Something that makes the user different and unique (e.g., fingerprint, retinal pattern, signature, voice, face, ...)

Based on at least one of them, an access control system can make its decision if the user can be authenticated and authorized to use a guarded resource. But this alone would still be insufficient to make any decisions. A policy would be needed in which is declared what is allowed and what is disallowed for a given resource and who are the users that can access it with what kind of access rights.

Furthermore a simple authentication and authorization system based on knowledge would still be insufficient to prevent for example a person called Bob to use a guarded resource which should be only accessible for Alice if he has the same knowledge as Alice. Therefore at least a two-factor authentication is used normally. This means that the key for the resource is still not enough to get access to it. A second credential would be needed, for example a biometric input. By adding such a second credential it becomes much harder that anyone else than Alice can access her resource.

2.5.1 Authentication, Authorization

Authentication is a process where the identity of a user and its given claim is verified. Authorization is a process where an authenticated user is verified if he has the permission to perform a specific operation or to access a specific resource. Therefore authorization cannot be done without authentication.

2.5.2 Access Control Policies

A typical access control policy specifies the resources, the users and their access rights on it. Shortly said it answers to the What, How, Where and When questions. This means what kind of users can how, where and when access a specific resource.

Widely used are XML based policies but proprietary formats are still used as well. The idea behind access control policies is to describe in a single file how a Policy Decision Point (PDP) should make his decisions for the specified resources based on which rules. Normally access control policies are defined to deny everything by default and only the additional added resources, users and their permissions on it are checked for the decision making process.

2.5.3 RBAC - ABAC

ABAC is the acronym for Attribute Based Access Control which means that the access to a resource is based on the attributes of a user. Such an attribute could be the number of available leave days to access the resource for booking a holiday. An example therefore is Permis [18]. In distributed grids where

resources are guarded by special attributes there is always the problem that all the attributes which are claimed by a user are probably not rightfully his, which means that it must be verified that a user can only claim attributes that he really owns. Therefore, a credential validation service must be introduced to validate and determine these attributes against the claiming user to decide if this subject is the real owner to gain access to the requested resource. Role Based Access Control or short RBAC is a specialization of ABAC. In the RBAC model, a role is not restricted to an organizational role. A role can be any attribute of a special subject like the current level of authentication (LOA) or a professional qualification. In a RBAC system permissions are never assigned directly to a user, only to a role. Every authenticated user can take therefore only roles for which he is authorized. Any transaction that happens is checked if it's done from a valid role and an authorized user that can take this role. Further a role hierarchy between the different existing roles is possible and mostly present. That means that if a higher role within the role hierarchy is linked with a user then the user has normally automatically access to all sub-roles of that role.

2.6 Trusted Computing

2.6.1 Overview

The term Trusted Computing is originated from the field of trusted systems. The idea behind Trusted Computing is to let the computer consistently behave in expected ways which will be enforced by hardware and software. The technology behind Trusted Computing is developed and promoted by the Trusted Computing Group (TCG) [43]. The TCG itself was formed in spring 2003 as a non profit industry standards organization and has adopted the specifications developed from the former Trusted Computing Platform Alliance.

The core idea behind Trusted Computing is to create a chain of trust. This chain of trust uses the Trusted Platform Module, which is described in the next Section, as the anchor of this chain. The idea is to have not only software elements in this chain which can be more easily broken than hardware modules which are only accessible through physical contact. The software modules in this chain of trust can be always monitored and be queried through the hardware module to ensure that only software is running which should. This query is called remote attestation if it is done from remote. A description about it can be found in the Section "Core Elements and Properties for Trusted Computing". Furthermore this hardware module utilizes RSA encryption. Every module owns its own private and public key. The private key never leaves the hardware module. These keys are used through the remote attestation process or indirectly for signing and binding operations, as described in the Section "Core Elements and Properties for Trusted Computing".

2.6.2 Trusted Platform Module

The central hardware for using the Trusted Computing technology is called Trusted Platform Module (TPM). The TPM itself consists of the following sub modules like shown in Figure 2.6.

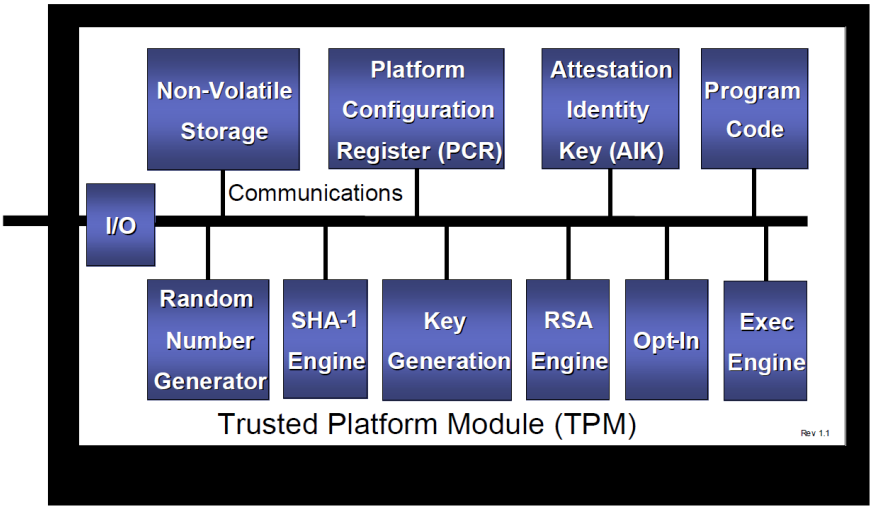


Figure 2.6: Structure of a Trusted Platform Module [43]

All communication with the TPM is done over the "Input/Output component" which connects all sub modules of the TPM. Therefore it is responsible for the appropriate fine grained transfer of all incoming and outgoing messages from and to the sub modules of the TPM. The "Non-Volatile Storage" is used to store the Endorsement Key, the derived Storage Root Key, the owner authorization data and persistent flags. The "Program Code module" provides the functionality for measuring platform devices. It represents the Core Root of Trust for Measurement (CRTM). A true "Random Number Generator" for

generating random numbers is present and is automatically used during key and nonce creation. Further it is used to strengthen the pass phrase entropy. The “SHA-1 module” inside the TPM is used for computing signatures, creating key blobs and for general purpose use. A fully autonomic “RSA Engine” for key creation, encrypting and decrypting can be accessed over the TPM too. The “Platform Configuration Registers” can be used to interrogate the current state of the platform. The “Key Generation module” is used to generate 2048 bit RSA signing and storage keys as specified through the TCG. To enable a preconfiguration of the TPM the “opt-in module” is needed. It allows to ship the TPM to the customer exactly in the state as required. This could be for example fully enabled or disabled and deactivated. The “Execution Engine” is responsible for the execution of program code. Further it performs the TPM initialization and the measurement of the configuration into the Platform Configuration Registers. The “Attestation Identity Keys”, which are derived asymmetric keys from the Endorsement Key, are located too in the TPM. The Endorsement Key represents a 2048 bit RSA key and never leaves the TPM. A detailed description about the Endorsement Key follows in the next subsection. For a complete TPM description look at [36] [43].

2.6.3 Core Elements and Properties for Trusted Computing

The fundamental elements to enable trusted computing are the endorsement key, the memory curtaining, the sealed storage and the remote attestation feature.

Endorsement Key

The Endorsement Key is normally created during the creation of the TPM chip and can not be changed. A creation of a new Endorsement Key is only on TPM’s possible which are based on software and not directly built into silicon. The Intel TPM is for example such a software TPM. The key itself is a 2048 bit RSA key and the private part of it never leaves the TPM. It is used to enable the execution of secure transactions by allowing the owner to show that he has a genuine TPM. To enable anonymity arbitrary many Attestation Keys can be derived from the Endorsement Key. These Attestation Keys can then be used for sensitive actions without using them twice to allow nobody to conclude what actions have been done from the same person. This Attestation Key can further be used for attestation and for encryption of sensitive data. For example, owners of an Infineon TPM can prove the presence of their hardware TPM with an certificate signed by Infineon for it. This certificate proves that a real hardware Infineon TPM module is used. The certificate itself can be verified through Infineon.

Remote Attestation

Remote Attestation allows the detection of configuration changes. These changes can be done in software or hardware. The Trusted Computing Group has released a set of specifications which allow dynamic attestation and to build an attestation agent. This agent can then be used to query the configuration of the target system and to create a certificate for it. This certificate can then be shown to a remote party to prove that unaltered software is currently be executed.

Memory Curtaining

Memory Curtaining [29] represents an extended form of host memory protection to provide a complete isolation of sensitive areas in the memory. The TCG specifications focus on protecting platforms against software attacks. AMD has therefore developed AMD Virtualization or short AMD-V [40]. Intel itself calls the same kind of technology Intel Trusted Execution Technology (TXT) [24]. If an intruder gets full access to a Trusted Computing system then the protected memory with the sensitive data is still

completely secure, because of the use of a separate protected execution environment. The operating system itself on such a system has only limited access to that area.

Sealed Storage

Sealed Storage is used to protect private data by binding the data to the platform configuration. This means that the current platform configuration is measured and based on this configuration the private data is encrypted. Only with exactly this configuration the private data can be encrypted again. If only a software or a hardware component changes than the measurement of the configuration will create a different measurement to the one before and therefore a decryption is no longer possible. Therefore the private data can only be released on exactly that configuration. This system can further be used to enforce Digital Right Management (DRM) for example.

2.7 Advanced Cryptographic Trusted Virtual Security Module

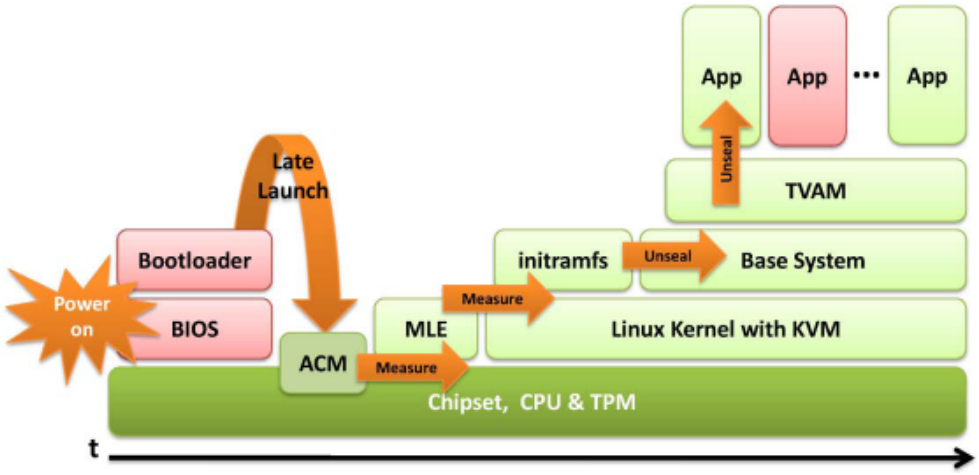


Figure 2.7: acTvSM: Overview about the Secure Boot [8]

The Advanced Cryptographic Trusted Virtual Security Module or short acTvSM system represents in the current version a prototype to utilize Intel's Trusted Execution Technology (TXT). The current version directly starts a Linux based operating system which has been especially modified to take advantage of all TPM based features and guarantees. Further it guarantees after the successful and complete boot process integrity for the booted system. Based on this base system any virtualized application can be hosted and started. Figure 2.7 shows the initial start of such a platform beginning with switching on the machine and ending with the successful start of one or more virtualized applications.

To achieve the system integrity the Bootloader (as shown in Figure 2.7) initializes after the chipset and BIOS initialization a so called Late Launch. During this Late Launch the system is reseted to a known and thrustworthy state although the Bios and the Bootloader has already been loaded. This secure state is guaranteed by the Chipset, the CPU and the TPM. All three of them are working together to reach the secure state. The Authenticated Code Module (ACM) resets all measurements, measures the current state and starts then the Measured Launch Environment (MLE). Measurement means that every software component is hashed and verified to ensure that the state of the component has not been modified. Only after the hashing and verification, which is the task area of the TPM, a software component will be loaded. This behavior is needed to protect the chain of trust. Further a Linux Kernel with Kernel-based Virtual Machine (KVM) support is started. The MLE measures during all further computations the platform state. Based on the successful start of the Linux Kernel the initramfs is booted which unseals the Base System and starts it. Finally the Trusted Virtual Application Manager or short TVAM is started. This manager is responsible for unsealing and starting all applications which are included within the acTvSM package.

Chapter 3

Grid/Cloud Computing Survey

3.1 Introduction

This chapter centers around BOINC, folding@HOME, Globus Toolkit and GridGain. The first two mentioned programs can be used out of the box without any modifications, which means that they are perfectly usable by any user. Simply start it and join the computational network to provide the network the local unused CPU power. The last two programs representing only frameworks to build grid and cloud computing applications and therefore they are not directly usable without further effort.

Through the following Sections the term server is always used in terms of the platform which is providing the workunits. These workunits are tasks which need to be computed to get the results. The clients are therefore providing their computational power to compute these workunits. Therefore the clients provide the computational services and the servers use them by sending them the workunits.

The reason why we will have a look into these programs and their architecutes is to analyse how these programs are dealing with the security issues. A detailed analysis how BOINC, folding@HOME, Globus Toolkit and GridGain are dealing with the main security problems, which have been mentioned in Chapter 1, can be found in Section 3.6 “Risks and how the above presented applications deal with them”. A gerneral view about the mentioned programs is included directly at the end of every program Section.

3.2 BOINC

3.2.1 Overview

The acronym BOINC stands for Berkeley Open Infrastructure for Network Computing [10] and represents an open-source grid computing platform. BOINC itself has been developed through the Space Science Laboratory on the Berkeley University. Its predecessor was the widely known SETI@home, which has been developed by the same department. Therefore, the SETI@home project was the first join-able project for BOINC. BOINC is further designed to allow any scientific researcher or group to participate with their own computational projects.

The general goal of BOINC is to provide a scientific computational environment for public users which want to participate in scientific research by sharing their private computational power. Private users can download, install and configure BOINC to participate on as much scientific projects as they want and they have the power to decide for which project which amount of CPU power, memory, storage and network bandwidth should be spent. BOINC is designed to be executable on a wide range of different machines and operating systems. It is completely open source and programmed in the language C.

3.2.2 Architecture

Any calculation that is done by BOINC is organized in projects. Everyone can create his own scientific project for BOINC. Therefore every project corresponds to an organization or a research group that is interested in public grid computing or at least in using public grid computing to speed up their computations. A project is always defined by a single master URL, which points to the root directory of its project home page. A user who wants to participate in a project has therefore simply to visit that home page with a normal browser or to enter the URL directly into the BOINC software and to register for it. Every project divides his computational work into small workunits that can be delivered to the users.

The BOINC server of a project is directly connected to a relational database. This database is responsible for storing all kind of information related to the project, like supported platforms, workunits, results, registered users, teams and more. The server functions itself are accessible through web-services and daemon processes.

The server services can be divided into scheduling services and data services. The scheduling service is responsible for the handling of RPC's from the clients, which means that it issues workunits out to the clients and handles the reports from uploaded completed results. The second one is the data service. It is in charge of verifying the result uploads. Therefore a certificate based mechanism is used to ensure that only legitimate files can be uploaded. The file downloads are done by plain HTTP.

BOINC is designed to add and manage more than one project, but only one at a time can be active. Any project and its workunits can be started, stopped and suspended as long and as often the user likes. Furthermore every workunit can have a deadline, which means that if no result for these workunits can be computed and uploaded before this deadline, then the BOINC client must reject these workunits.

BOINC is available as a direct executable binary file (32 bit and 64 bit-versions) for Windows, Linux and Mac OS. For other platforms the source code can be downloaded, configured and compiled to use it there too.

After the download of one or more workunits the BOINC client can run the computation process completely without any internet connection. Only for the upload the connection is needed once again. Nearly every feature of BOINC can be configured by the end user like how much CPU resources, local memory, storage or network bandwidth should be used at all. Moreover BOINC allows configuring under which circumstances it should do its calculations like only when the CPU is completely underemployed or only between well defined time frames. These settings can be done too by machine and or by user account if a user shares more than one machine to compute.

3.2.3 Structure Analysis

Workunit Structure

Every project for BOINC has to define application versions for the different supported platform configurations which should be able to compute for this project. Every application version is then very precisely defined for each architecture like Windows 32bit with an Intel CPU or Linux 64bit with an AMD CPU. Every application version contains a set of files that are needed on that kind of platform configuration to perform the computations. These files for the specific application version will then be transferred to the client during the first connection with the BOINC client to the specific project web server.

Every workunit for a specific project is defined as a set of input files, a set of command-line arguments and a set environment variables. Further it includes a deadline for every workunit until the result must be uploaded. After that deadline the result of the workunit is not longer valid and therefore it can not be uploaded anymore. Detailed information about computing time, storage and memory usage are included and needed by the local BOINC client to verify if this workunit can be computed on the local machine. Every result package contains a reference link to the source workunit and a set of the created result files during the computation process.

All file names are unique and immutable within a project. But the files can be replicated for sending the same workunit to different users. Therefore every file name on the server is associated with a list of referenced workunits, application versions and results.

Redundant Computing

The problem with incorrect results from malicious users is handled by BOINC's redundant computing logic, which means that one workunit is sent to different users over a defined time frame. If homogeneous redundancy is activated for a project then one workunit is only sent to users with the same architecture, operating system and CPU vendor to overcome the problem of slightly different results because of different operating systems or architectures. These slight differences are based on the usage of different mathematical libraries and their different implementations through the different operating systems. The CPU architectures itself are normally not completely bug free [28] and therefore different roundings of the decimal places or other incorrect calculation steps are possible.

Failure Management

Because anyone can host his own BOINC project, a lot of them are hosted on low-end servers. If only a small number of users connect to such a server from time to time then in most cases there will be no delay for the users, but if thousands of them try to connect to the same server then it will come to a denial of service. To prevent this, BOINC automatically takes an exponential communication break for this project server. This means that the BOINC client will not immediately try to reconnect to the same project server again. The delay time before the next try starts depends directly from the number of errors that have already happened with this host. If no workunit at all is left for this project then BOINC will switch to another project if available.

3.2.4 Participant Motivation

BOINC gives credits for any workunit done. The number of credit points can be different for any workunit and project, because they are defined as the combination of needed CPU power, storage usage, memory usage and network bandwidth usage. These credits are more or less only a status symbol because nothing can be bought with it. Only from time to time a cup or a t-shirt can be ordered if enough credit points have been accumulated. These credit points are more used to verify for the users which rank they currently have. This means they can check which position they hold in the statistics for their country or city for example. Or which person has the most credits but only computed with one computer. The idea behind these credit points is to motivate the users to have a competition with other users about their amount of credit points.

Another motivation but with the same background is the feature to build teams. These teams collect Credit points as well and they have their own statistics. But the biggest motivation for the users is maybe the screensaver option. This feature allows the BOINC client to show in real time (if supported through the project) a graphical representation of what is currently computed while the computer is unused. If the project doesn't implement a visualization, then BOINC generates by itself some screensaver content, which isn't in any relationship to the current project.

Further any user can decide the computation power for each project he has joined. This allows a user to decide which project should gain more computation power and which one less or no computation power at all.

3.2.5 Security Issues

Boinc uses a replication system for the problem that a malicious user tries to upload wrong results or to fake his amount of credit points (by claiming more CPU power for a workunit than he has really needed). This system sends one workunit at least to 3 different users during an specific time frame. If the result upload and the credit claim is the same from all the users then BOINC takes this result as a correct one.

Every new project must, before it can be spread to the users, be registered and signed by BOINC. This should guarantee the users that only non-malicious code is distributed. But because it is not public how BOINC checks any new project-code before they sign it, only the asumption can be made that they are looking in detail into the code. Every workunit for a project itself can then be signed by the project to verify that a workunit really originates from that project server.

Preventing denial of server attacks on data servers is an optional feature of BOINC. Every result file has its maximum size and for each project an upload authentication key is present where the public part is directly stored on the project's data server. The file descriptors for the results are already signed with the private part of the project server key. These file descriptors contain the maximum sum of data that the result upload is allowed to have. The users receive these file descriptors from the project server, combine them with the results and upload them to the data server. The data server recognizes the signed file descriptors and accepts therefore only files within the size range defined by the file descriptors.

Against theft of participants' account information by server attack or by network attack BOINC has no security features to offer to prevent this. This means that against attacks of BOINC project servers only the local project group is qualified to ensure that the server is secure. Further all the project files are saved unencrypted on the participants hard drives. The general purpose for this is that the data resides as well in plaintext in the memory.

The communication between the participants and the project servers is done via HTTP. But since 2008 the clients are supporting the HTTPS protocol too.

3.3 folding@HOME

3.3.1 Overview

Folding@HOME is similar to BOINC but it only supports scientific research in the sector of protein folding. The idea for folding@HOME was born in 1999 by Vijay Pande at the Stanford University [12]. Within a short time frame folding@HOME was available as a BOINC project too but because of security reasons the project team decided that only the unimportant parts of the folding@HOME source code should be available. Therefore they developed their own client software and canceled their public project for BOINC. These unimportant parts are the GUI, the graphical presentation of the folding process itself and the interface for using GPU's for the computation process. Another reason why folding@HOME doesn't share any other source code maybe the reason that the protein folding market is highly competitive and the used algorithms are worth much. Nowadays participants can only participate at folding@HOME with their client software. The biggest challenge for folding@HOME was the circumstance that their computations (in the beginning) were scaling very badly. The smallest workunits from that time have needed more than 8 hours of computation time to get solved. Therefore a lot of work had been invested to develop proper algorithms to scale the computations down to small pieces. The main goal of folding@HOME is to provide a scientific platform for medical protein research for public participants. Folding@HOME can be downloaded for various platforms and architectures like Linux, Windows, PlayStation 3 (PS3) or Mac OS but there is no open source version which could be compiled on an alternative platform.

3.3.2 Architecture

Similar to BOINC folding@HOME allowes different projects. All these projects are centered around protein folding. A user joins automatically a random project or if he knows a specific project number then he can join a specific protein folding project as well. But in general a user does not decide by his own for what his computation power is used. There are different types of workunits and not all of them can be computed on any platform. For example a typical workunit for the PlayStation 3 (PS3) needs nearly 8 full hours to compute. Because of the support of Sony, all PS3 models are equipped with the folding@HOME software right away. Only if an update arrives then the user has to start the update manually. The main feature of using PS3's is the enormous power of the Cell processor. In 2008 the speed was up to the 27x of an Intel core2Duo processor with 2.4Ghz [41][16].

There is a wide range of adapted program versions for conventional personal computers. They distinguish not only in different versions for 32-bit and 64-bit, but further in models supporting graphic cards, like ATI and NVIDIA, for doing the calculations. Nowadays graphic cards like NVIDIA and ATI can be directly used via CUDA [6] for NVIDIA based cards or OpenCL [3] for both kinds of cards to do computations directly on the graphic card processor. Using these cards for the computational process brings the benefit that they are a lot faster than conventional CPU's. But because that they are originally graphic cards not all kind of calculations can be done and therefore only selected workunits can be computed on them.

Compared to BOINC only a few configurations can be done by the participants like allowing bigger workunits or how many CPU's (on a multi-core system) should be used.

3.3.3 Structure Analysis

Workunit Structure

There are 2 main types of workunits: non-SMP (default) and SMP, where non-SMP workunits must be calculated on one single core and SMP workunits can be calculated on as many available CPU's as

possible. A multicore system can therefore do one non-SMP workunit per processor. Furthermore the workunits are digitally signed with a 2048bit RSA key. If folding@HOME detects workunits which are not correctly signed then the client throws them away and tries to download new workunits.

Malicious results

In the case of folding@HOME, a function is called from time to time directly at the client-platform which checks the numerical correctness of the computing system [42]. This means that some built-in tests validate the always fresh computed results against the known results. More than that check for correctness is not done for the folding@HOME workunits. Because of the circumstance that the workunits need a lot more time than the conventional workunits for BOINC no redundant computing is done. Furthermore the client base is much smaller than the user-base that BOINC can use and therefore one workunit is always solved by one user.

Failure Management

If the folding@HOME client recognizes computational error for a given and active workunit then it drops the workunit and downloads a new one. Before the folding@HOME-client is able to download new workunits, the signature of the used folding@HOME client is verified against the folding@HOME project server. If the signature is no longer valid then a new folding@HOME client must be downloaded before any new workunit can be downloaded again. In the case of the PS3 client, no direct download is available. Only an automatic update directly over the PS3 operating system can be initiated and therefore no self-manipulated client can be installed. All in all these are lightweight strategies to overcome the risks of public grid/cloud computing which can be found in the Chapter 3.6.

3.3.4 Participant Motivation

To motivate the participants folding@HOME uses roughly the same system as BOINC. The participants can access the same kind of statistical material as within the BOINC project. The calculation of the points differs from workunit to workunit and there are different ways to earn bonus points. For example if a users calculates a workunit within a day he gets a bonus, or if he computes more than x workunits a week or 24 hours a day he earns bonus points.

One advantage against BOINC is that the screensaver always shows the current status of the computation of the protein folding process. This means that a participant can always quite exactly see what he is currently computing.

3.3.5 Security Issues

In the case of folding@HOME all workunits and the client software itself are digitally signed by folding@HOME. The download of the client software and the workunits are only available from there. For the signing process an asymmetric RSA 2048 bit key is used. For preventing malicious workunits the client always verifies that the workunit is properly signed.

The client software itself is only available as executable files for the different supported platforms. Only parts of the software are open-source like the module for the graphic presentation [46] or the source code for using GPU's [46] for the computation. All the logical and the core algorithmic parts of folding@HOME are closed source and not open to public viewing. This is well-grounded by the claim of the intellectual property of the developed algorithms.

The security risk of theft of calculated results is not handled at all.

To prevent a denial of server attack during the upload of the results, the upload server accepts only results from signed workunits and signed clients. The client itself measures itself during the start and later during the computation from time to time to verify that the computation is correct. Information about this measurement and how it works exactly are classified and if the checksum is not correct then the upload server rejects the data from the maybe “malicious” client.

In contrast to BOINC, folding@HOME does not send the same workunit to more than one client to recognize calculation errors or to detect malicious users. Thus folding@HOME uses only the workunit and client signing feature and during the computations the calculation checks.

3.4 Globus Toolkit

3.4.1 Overview

The Globus Toolkit [21][47] represents an open source grid computing framework technology. Its first release was already in the year 1998 (GT 1.0.0) and the latest release is actually the version 5.0.1. The development is lead by the Globus Alliance, a consortium based on the University of Chicago, the Royal Institute of Technology of Sweden, the National Center for Supercomputing Applications, the Univa Corporation and the University of Edinburgh.

The idea behind the Globus Toolkit is to present a framework for grid computing with components for fault detection, security, data management, resource management and communication. Instead of reinventing the wheel, the Globus Toolkit tries to use only well developed standards for its services, as SSL/TLS, SSH, LDAP, X.509, SOAP, HTTP, GridFTP, XACML, SAML, WSDL, OGSI and many more [47].

The Globus Toolkit is developed in C and its source code is available for Mac OS, Linux derivates, NetBSD derivates and Solaris.

3.4.2 Architecture

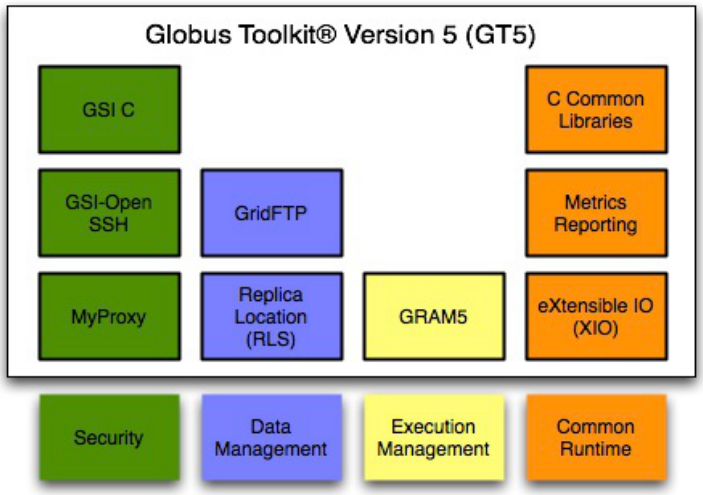


Figure 3.1: Components of the Globus Toolkit Version 5 [47]

The design of the Globus Toolkit is completely modular (as shown in Figure 3.1). All of its components can be used together or completely autonomously. The existing modules of the Globus Toolkit can be divided into one of the four main types Security, Data Management, Execution Management and Common Runtime. These main types include the following modules:

- GSI C: This module provides API's and tools for authentication, authorization and certificate management for web services. The authentication API is centered around a public key infrastructure and allows the usage of TLS and certificates as the X.509. Delegation mechanisms are supported through the authentication API as well. For the authorization process are two modes available: The first one allows granting access rights based on the client's credentials and the second one based on the user names.
- GSI-Open SSH: This module represents an adapted version of OpenSSH. The special feature is the added support for X.509 proxy certificate authentication. Further delegation, single sign-on

remote login and a file transfer service has been added. The main task for this module is to allow remote login and file transfer between systems without entering a password.

- MyProxy: The MyProxy module manages the X.509 certificates and private keys. It combines an online credential repository together with an online certificate authority. By using this module users are able to authenticate and to obtain credentials.
- GridFTP: This module is based on the widely known file transfer protocol (FTP) and it is responsible for secure and reliable data transfer. GridFTP not only includes the standard features of FTP but much more a lot of features specialized for reliability, security and more data transfer performance. Some of the added features are for example the failure restart capability which allows to store a list of not complete transferred files for a later restart or the stall detection which allows to detect and to handle transfer errors.
- GRAM5: GRAM5 is the acronym for Grid Resource Allocation and Management. Its core intention is to monitor jobs and to allow to locate, to submit, to monitor and to cancel them. This whole module is based on a set of services for the different kind of tasks.
- C Common Libraries: This module combines all the needed C libraries for the different services of the Globus Toolkit together in one package and adds an abstraction layer above them for an easier and controlled access.
- eXtensible IO (XIO): This last module represents an extensible input/output library which is accessible through only one API. This API supports a wide range of protocols like TCP, UDP, file access, HTTP, TELNET, queuing and much more. With intent this API is designed to support the four basic functions for open, close, read and write on all kinds of the underlying usable protocols.

3.4.3 Usage Motivation

The Globus Toolkit represents a completely different framework against BOINC or folding@HOME. It can not be used out of the box for grid/cloud computing but much more it allows with its modular design to create a powerful and own grid/cloud application. The Globus Toolkit is nothing for an end user (in its unmodified version) but much more for developers which want to get a powerful solution with already included modules for all kinds of grid/cloud computing. Further it is completely coded in C which makes it very fast. Another feature is the commercial support trough the Univa Corporation if needed.

3.4.4 Security Issues

Because of the difference to BOINC and folding@HOME in the matter that the Globus Toolkit is representing only a framework to build an own grid/cloud application, all the security issues are more or less in the hand of the developer which forms the framework to an usable application. This means that the developer is responsible for the final outcome how secure the built application is. There are already a lot of security-features included which can be more or less easily and directly used.

The security concept of the Globus Toolkit is based on public key cryptography. Further it includes tools for creating and using an own Certificate Authority through the SimpleCA implementation which is able to issue X.509 certificates. For communication a modified version of OpenSSH can be used for secure, fast and reliable remote connections.

Against the problems that have been mentioned before in the BOINC and folding@HOME Section 3.2 and 3.3 the developer itself is responsible to use the given tools to protect the sensible data in his grid environment. The choice to use the programming language C may not be the best one in the matters of buffer overflows, type safety and memory leaks. Nowadays many security applications are developed in Java to use the language specific security features.

3.5 GridGain

3.5.1 Overview

The GridGain framework [30][27] offers an open source grid and cloud computing framework based on Java and for the Java language. The first version of GridGain has been released in February 2007 and since that it has been continuously developed to its current version 2.1. GridGain itself is dual licensed under the LGPL and the Apache 2.0 license. On first view GridGain looks more or less similar to the Globus Toolkit framework but it has a lot of specialties to offer that are different. One of these specialties is that every existing Java code can be adopted to run on this grid/cloud computing framework in only a few seconds. Only the fine tuning to use the grid/cloud environment more efficiently with existing code could take a little bit longer. The next difference is that GridGain runs out of the box on every Java capable platform without any additional effort.

Like the Globus Toolkit GridGain is open source but for commercial needs, paid support is directly available through the developers. The main goal behind GridGain was to design an easy to use grid computing environment, but it has been developed to a cloud computing environment as well. Cloud computing environment means that the end-user is no longer aware that he is using a grid service. Therefore the cloud computing layer, which is above the grid computing layer, covers completely the underlie grid computing services with its own services.

GridGain is less powerful than the Globus Toolkit in terms of security features but with the background of using Java much more secure in the field of type safety or memory leaks for example. Furthermore GridGain includes no kind of authorization/authentication features or security features to prevent unknown nodes to connect to the GridGain server.

GridGain itself can be downloaded as Java source code or as an installable executable binary file for Windows, Linux and Mac OS.

3.5.2 Architecture

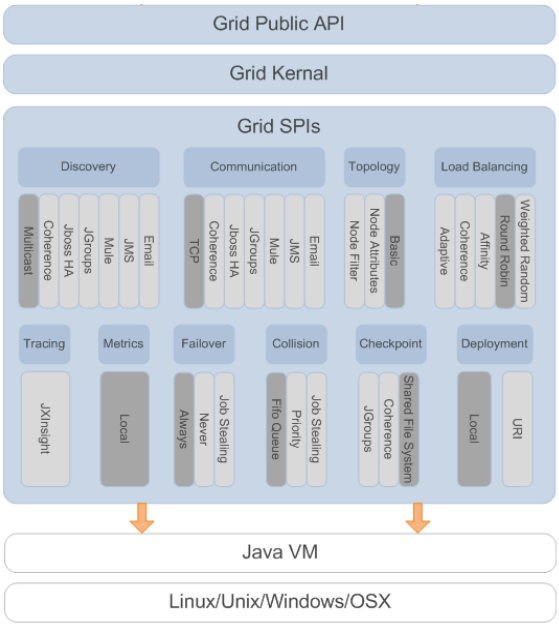


Figure 3.2: The layered structure of GridGain [27]

The architecture of GridGain is based on layers (as shown in Figure 3.2). The highest layer is called Grid Public API and it represents the interface layer which is used for all kinds of interactions

with GridGain. The developer/user uses this API for executing his application, for getting grid specific information, for configuring GridGain to his needs and for accessing all the other features of GridGain.

The GridKernal represents the middleware between the public API interface and the Service Provider Interface (SPI). The name kernal was chosen explicitly in honor to the old 8-bit Commodore operating system by the developers. The main task of the GridKernal is to implement and to provide all necessary functions to fulfill all the public services provided in the Grid Public API and to delegate them to the deeper Grid SPI layer.

In every package of the shown SPI's in Figure 3.2 one sub entry is always dark gray. This is the default selection if nothing has been changed in the GridGain configuration file or nothing has been directly selected during the start of a GridGain node.

The SPI is the true gateway to the GridGain functionality. It is responsible for all public features of GridGain and it is splitted into the following sub parts:

- Discovery: The Discovery package delivers the functionality to discover new nodes. Therefore a bunch of different discovery modes have been already implemented and they can be chosen easily and directly by changing some lines in the configuration file or by creating a Java GridGain configuration object and change it there directly. As shown in Figure 3.2, there is a wide range of different discovery modes available for GridGain like for example via multicast, via e-mail or via (E-)Mule.

- Communication: This package is quite similar to the discovery package but it is responsible for all kinds of communication over the selected communication way like for example via TCP or via E-Mail.

- Topology: A topology can be restricted to contain only nodes with special attributes or with other unique features. The Node Filter Topology can therefore be used to add for example only nodes that have at least 4 GByte free memory and 2 free CPU's available. The Node Attributes Topology is similar to the Node Filter Topology but it can select the nodes for a given topology by its given individual attributes and not by its given system information only.

- Load Balancing: This SPI is responsible for all kinds of Load Balancing. GridGain has already 5 implemented and available different versions of them to offer. Generally if nothing is chosen then by default the Round Robin load balancer is used.

- Tracing: This package allows to monitor the local used topology with the available JXInsight Java tool. This application offers tools for monitoring, problem diagnostic, transaction analysis and application management. More information about JXInsight can be found at [5].

- Metrics: This class offers different getter-functions to query all kind of available information about the current selected node like available processors, average number of active jobs, average number of waiting jobs, current CPU load and many more.

- Failover: The Failover package is responsible for the task what should happen if something goes wrong with a job. There are different kinds of included solutions for these scenarios starting from just do nothing, do the job again and ending by give the job to another node.

- Collision: This package looks quite similar to the failover package but it is not. It is responsible for the proper handling if two or more workunits are arriving at the same moment on the same node or if there are already some workunits waiting for being executed and therefore for that kind of job execution handling.

- Checkpoint: For larger workunits it is quite useful to use execution checkpoints. With them it is possible to start, stop and suspend workunits without losing already calculated data. To enable

this kind of feature it is necessary to add the proper needed methods directly into the code of the workunits.

- Deployment: The last package is the Deployment package. It is responsible for all kind of deployment of the workunits and their additional needed data.

3.5.3 Usage Motivation

Especially for Java developers it is very easy to adapt existing code for grid/cloud computing with only adding some annotations. The next big advantage against the Globus Toolkit framework is its instant runnability on every Java capable platform without any modification. For the first steps or a simple HelloWorld program an inexperienced developer will need only 15 minutes [26] and this is simply the best starting motivation that a developer can get. But GridGain itself can be very easily configured and be adapted to special needs. It offers a lot of different grid computing settings that can be switched in seconds simply by starting the GridGain node with different parameters. Another motivation to use GridGain instead of another grid framework could be its simplicity and clarity in the matter of its usage and how to develop within the GridGain framework. Further GridGain includes a lot of runnable examples to show how it works and how powerful it is. Only the missing security features are an obvious weakness of GridGain.

3.5.4 Security Issues

GridGain is simply not designed to protect itself against malicious users. It is designed to develop grid/cloud computing applications in the same way as developing stand alone applications. It offers features for real time deployment of code to other GridGain nodes and to do load balancing in every needed situation. Furthermore it has features for fault-tolerance but it can not really handle malicious results from one or more bad nodes. That is the reason why GridGain can't be used for any security based application without modifications.

3.6 Risks and how the above presented applications deal with them

The shown grid/cloud computing frameworks are offering a lot of possibilities for successful and efficient grid/cloud computing. But how secure are they in the matter of data protection, malicious participants, data theft or malicious results? A recent report by ENISA [15] lists for example a whole bunch of existing different risks for cloud computing networks and the overall statement is that many of them are still completely unsolved and therefore much more research must be spent into this scientific part. In the paper "Innovations for Grid Security from Trusted Computing" [31] a lot of the same difficulties within grid/cloud environments can be found. Therefore we will analyze the just reviewed programs against our main security list which is directly taken from [49].

1. Participants may deliberately report back wrong answers.

   In the case of BOINC, this problem is simply solved by sending the same workunit to at least 3 users and then to verify the results if they are completely the same. If not then some more users will have to compute this workunit until it is clear which result is the correct one. Not very efficient and if the BOINC workunit scheduler chooses the same kind of malicious users which compute for example only with single precision than double precision then a faulty result will be uploaded and accepted by the BOINC server.

   Folding@HOME does not handle this kind of problem at all.

   For the Globus Toolkit, the developer who fits the different modules together is responsible how well it can resist against that kind of attack. But without a chain of trust which is not only based on software, code manipulations are hard to detect.

   GridGain has to deal with exactly the same problems but different to the Globus Toolkit it has less included security options to offer which can be activated. Only by using some external Java components this kind of risk can be reduced by a little but not completely because code manipulation is still hard to detect.

2. Participants may not enforce data integrity or not compute accurately.

   As mentioned above, BOINC handle this kind of risk by calculating every workunit at least 3 times to ensure that the computation was correct. But at all this precaution is not able to handle this risk completely.

   In the case of folding@HOME the client software computes during every start a hash sum of its executables and transfers it to the workunit server. Every workunit is combined with a hash sum. During the calculations the client verifies from time to time the numerical correctness. But if a user changes the client to not perform these numerical correctness checks then it is possible that faulty results reach the upload server. Further does folding@HOME not send one workunit to more than one user because of the reason that the workunits are much bigger and the user base is much smaller than the user base for BOINC. Therefore a faulty result can be uploaded and is hard to detect.

   For the Globuts Toolkit and GridGain it is exactly as mentioned before. The developer is completely responsible to handle that kind of risk.

3. Data protection requirements of sensitive data cannot be enforced on remote systems.

   In the current verion BOINC does not have any instruments for protecting its workunits on remote systems. All data is stored unencrypted in the computer memory and its storage and is therefore completely unsecured.

   Folding@HOME handles this kind of risk completely in the same way as BOINC.

For the Globus Toolkit and for GridGain again the developer is responsible to secure sensitive data. But both frameworks do not include such security features. Therefore this risk can only be handled by developing an own solution.

4. Theft of intellectual property on the distributed code by the participants is possible.

   BONIC is completely open source and therefore the used intellectual property is public to anyone. The structure of its workunits is public as well.

   Folding@HOME tries to protect its intellectual property by only making unimportant parts of its code public. The workunits are in the same way unsecured as in the case of BOINC.

   For Globus Toolkit and GridGain the developer can try to secure his important code by using software which changes all names of the functions, the variables and the constants to rubbish-names. This would ensure that with simple code analyzing tools a regeneration of the compiled code is not easily possible. But with up-to-date code analyzing tools it would still be possible to analyze the exact behavior of each function and therefore it is not possible to secure the intellectual property.

5. Theft of confidential data by the participants is possible.

   In the case of BOINC everything is unsecured. All the data is transferred, stored in the memory and on the hard disk unencrypted. Therefore it is completely open against that kind of risk.

   Folding@HOME does not protect the calculated data and therefore it is completely insecure similar to BOINC.

   In the Globus Toolkit environment the security depends strongly on the developer how he secures his framework. At least the transfer of the workunits can be easily secured by using OpenSSH for the transfers. But if someone has physical access to the client-platform then the theft is still possible.

   For GridGain it's again completely the same handling of the risk like above in the case of the Globus Toolkit.

As shown the current solutions do not or only partially secure against that kind of risks. Therefore a more secure and trustworthy grid/cloud environment is needed to overcome at least some of these risks.

Chapter 4

An Integrated Solution for Trustworthy Grid/Cloud Computing

4.1 Overview

This chapter describes very briefly the three used software components with all necessary details like their features, their application areas and their weaknesses. Section 4.5 will then center on the design process of our new framework and how the three base components must be adopted to fit properly together. The idea behind this new combined framework will be in the first step to extend GridGain with an authentication and authorization feature to achieve that only authenticated and authorized users can join our grid/cloud computing network. This is our first new security feature that we add. Further with a PKI behind our new framework, which will be described in Chapter 5, a more trustworthy grid/cloud computing network can be created than the shown grid/cloud solutions in the Chapter before. Because of the multiplicity of issues and failures that can occur only during the installation of the first two programs a detailed description of any necessary step to get those programs working can be found in the appendix. During the second step, which can be found in Chapter 5 as well, we introduce with help of JSR321 trusted computing into our new framework. This will be our second important change to the existing framework. That means that with the help of the TPM and Intel TXT we are able to add a hardware based chain of trust. This chain of trust can then be used to show for every client-platform that it runs in an expected state, before it can join our network.

4.2 GridGain

The list of GridGains key features is very comprehensive. Only one thing raise concerns, the security issue. In its basic design there are no security options to prevent the grid/cloud network against any kind of intruders or malicious users. Of course GridGain can be configured to take only nodes with special attributes but this is really a very low boundary to keep intruders out of the network.

As already shown before in Section 3.4 the features of GridGain are accessible through its public API. The middleware, the GridKernal, is therefore only responsible for connecting the API with the different underlying modules from the Grid SPI (Service Provider Interface) layer. The idea behind this layer is to enable a modular system. For every core module as communication for example different implementations already exist. Furthermore it is possible to implement own modules and to use them.

Therefore a deeper look into the Grid SPI has to follow to show some of the noteworthy features of GridGain in detail:

- Discovery SPI:

The discovery SPI is responsible for the discovery of new nodes and the detection of failed nodes. Independent from the used implementation of this interface all implementations are offering at least the functions to find a node by its UUID, to ping a node by its UUID, to receive a list of all found nodes, to get the local node, to add and to remove a discovery listener.

- Communication SPI:

  The Communication SPI provides all needed services to send and receive messages between all known and fully registered grid nodes. Dependent on the chosen Discovery SPI implementation, the same underlying network structure is used for the communication as well. GridGain itself defines two kinds of messages, one kind for the task execution and another one for querying and monitoring other nodes. These control messages are normally higher prioritized than the normal messages for task deployment and execution. The three basic functions, with all implementations for this interface must support, are the ability to send messages, to add a message listener which waits for incoming messages and a function to remove a message listener.

- Collision SPI:

  The Collision SPI is responsible for regulating the tasks in the grid and to define how they are executed after they have arrived on a certain node. This deals with the situation that normally on a grid node multiple tasks are arriving for execution or already waiting for being executed. In this case the Collision SPI has to deal with the question in which order they have to be executed: in parallel, in sequence or based on a time stamp which decides which comes first or maybe that only a certain amount of them can be executed and the others must be sent to other nodes.

  Further it implements the feature of task rejection and task cancellation. The difference between them is that rejection means that a task is aborted before its execution and cancellation means that a task is aborted during its execution. Further it offers such features like job stealing (shown in figure 4.1). Job Stealing is a feature that allows underemployed nodes in the network to steal jobs from busy nodes. In the case of figure 4.2 the free node 3 takes the Job13 from the busy node 1 and the Job21 from the busy node 2 to fill its own Job queue. If activated, this feature enables fair job spreading over all available nodes.

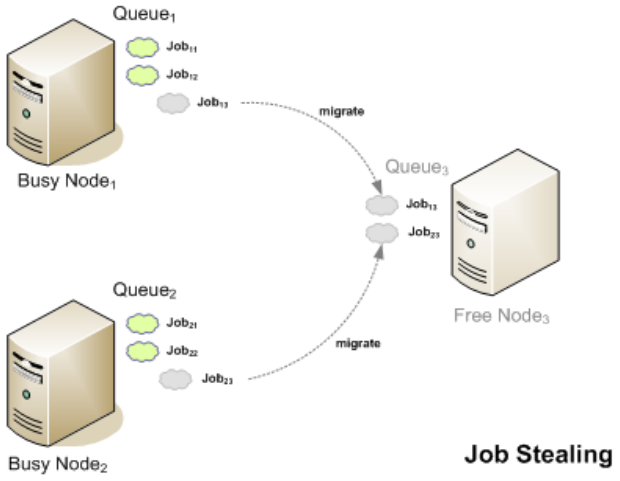


Figure 4.1: GridGain: JobStealing [27]

- Failover SPI:

  For every task a new Failover SPI implementation can be chosen. This allows to define fine grained exception handling for any given task. This exception handling only occurs if something goes

wrong during the remote execution of the task like for example the remote called task calls an exception, a task returns a bad result, a node on which the task was running crashed or the task was simply rejected from the remote node.

- Topology SPI:

  Similar to the Failover SPI allows GridGain a new instance of the Topology SPI for every new task to be used (one per task). This means that every new task can be equipped with its own Topology SPI instance. For example this feature can be useful if different tasks need to be executed on different platform configurations. Therefore this feature can be utilized to execute specific tasks only on platforms which fulfill specified requirements. The Topology SPI delivers a developer all kinds of information about the present nodes in the grid. Furthermore a developer can use this to perform tasks only on nodes with specified attributes. These attributes can be system attributes or manually set attributes by the user. The system attributes are containing all kind of information about the architecture, the operating system, the used software components and all set variables from the classpath. The manually set attributes allow to individually add further details which can be used on the server grid node for a more detailed splitting of the existing nodes.

- Load Balancing SPI:

  The Load Balancing SPI is responsible for choosing always the best node for the next task deployment. But how to choose the next best node if data partitioning is used? Data partitioning means that needed data for the next task execution can not be found directly on one node. The left side of Figure 4.2 shows this more complex example. Processing Node 1 gets the next task. In step 2 the needed data for the execution of this task is transfered from the Data Nodes 1 and 2 to the Processing Node 1. Then finally in step 3 the Processing Node 1 can start with the execution of the given task. After the result is computed it is sent back (step 4).

  The right side of the Figure 4.2 shows the feature of using data partitioning with Affinity Map/Reduce. Affinity Map/Reduce represents a built-in feature of GridGain. This feature allows a developer to design tasks which can be divided into sub-tasks. The idea behind these sub-tasks is to design them in a way to let them be executed directly on one node without the need to request data from any other node. This allows the task receiving node 1 to split the given task into 2 sub tasks and to deploy both sub tasks to the data holding nodes 2 and 3. The sub tasks can now be executed directly on the data holding nodes and the sub result is then (in step 3) sent back to node 1. Node 1 aggregates then both sub results and sends the full result back. With this feature unneeded data copying from other nodes can be completely avoided.

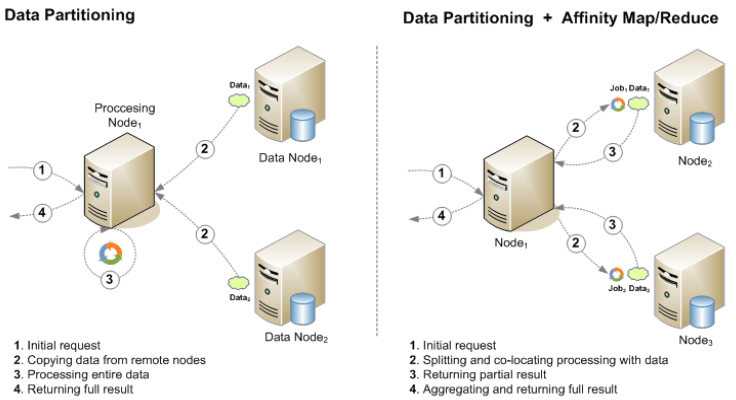


Figure 4.2: GridGain: Data Partitioning versus the GridAffinityLoadBalancingSPI [27].

- Checkpoint SPI:

This SPI enables the feature to suspend a job by storing its intermediate job state. This feature can be quite useful if the task needs long computation times and the nodes need a restart from time to time for various reasons. The specialty is that a suspended task can be transferred to another node and the execution can be continued there without any loss.

- Deployment SPI:

  The Deployment SPI is in charge of the deploying of the given tasks to the already chosen nodes by the Balancing SPI.

- Tracing SPI:

  The Tracing SPI allows to trace all kinds of events within GridGain like collecting statistics about nodes, the search for patterns or simply to monitor a specific node. To take advantage of this SPI it is necessary to use the external JXInsight Java tool which can be manually downloaded from the JXInsight homepage [5].

As shown above GridGain is very powerful and modular. For every SPI an own implementation can be developed as well. Only true security features for making the network topology more trustworthy and secure are completely missing and this is one of the biggest disadvantages of this framework. Therefore we will combine GridGain with Permis and JSR321 to improve the security of our new to design grid/cloud computing framework.

4.3 Permis

Permis represents a policy driven RBAC infrastructure [18]. The main advantage of policy based authorization infrastructures against hard coded systems or access control lists is the ability to be more flexible, scalable and application independent. Further they normally include tools for creating policies, managing the user privileges and deciding how the control decisions should be made. Therefore different types of policies may be used like access control policies, delegation policies, credential validation policies or credential issuing policies. In the case of Permis all these separate policies have been formed together in one policy. This feature of only having one policy is relatively new and exists only since the latest version. The idea behind these policies is to describe very precisely rules and criteria under which circumstances a user has which privileges or credentials. The user credentials are normally digitally signed to protect them against all kind of modifications and to guarantee their integrity. Based on these policies all access control decisions can be made.

Especially in the field of grid/cloud computing which is running over multiple domains, such policy based authorization systems have a lot of advantages to offer. They have the ability to control only the issuing of credentials in one domain and allowing the user to delegate autonomously their privileges to other users. Further these users have the ability to validate each credential of the resource domain and they have further the possibility to decide which users they trust and which one they don't for everyone separately. This kind of feature can not be found in the most existing grid environments and frameworks today.

Therefore Permis is a good choice to add these kinds of security and trust features into our new grid/cloud computing framework. Furthermore we will use Permis in our new grid/cloud computing framework only as solution for our authorization process, which will be done whenever a new node tries to connect to our network.

Permis itself is formed by the five following parts which are needed to make its access control decisions: The Policy Management, the Credential Management, the Authorization Decision Engine, the Policy Decision Point and the Credential Validation Service. Reasoned by the fact that a description of the operating mode of Permis is to far away from the scope of this thesis and the fact that we will not modify Permis at all, we will not describe these modules and their correlation to each other now. A detailed description can be found in [18].

Our main-goal will be much more to access Permis directly out of our GridGain framework and to configure Permis properly. For the configuration we will use the Permis Policy Editor, which will allow us very easily to create and maintain the proper policy files. A very detailed description about the modifications to the GridGain framework to enable a direct access to use Permis as authorization tool can be found through Chapter 5 theoretically and Chapter 6 practically.

4.4 JSR321

JSR321 [45][35] is used as an acronym for Java Specification Request number 321. The idea behind this project is to integrate Trusted Computing (TC) functionality directly into Java. There exists already a first reference implementation for this specification that can be used to get TC functionality within the Java environment. For the usage of TC under Java or the JSR321 reference implementation (RI) a Trusted Platform Module (TPM) is needed which could be found through a wide range of different hardware platform vendors.

The features of JSR321 are its high level API, the relative easiness to understand the various functions and that it only offers a really small set of functions. The point that JSR321 is completely available in Java allows us to integrate it directly into our new grid/cloud computing framework. Furthermore allows the integration of trusted computing into our new framework to establish a hardware based chain of trust and to use the TXT feature of Intel to ensure that all our client platforms are running within a known and good system state. This validation is made by using the remote attestation feature of trusted computing.

All in all JSR321 gives us the final instruments to extend our new grid/cloud computing framework with different security features, like the known system state or the hardware based chain of trust. That is the reason why we will take now a deeper look into JSR321 to see the different available features and where they are located.

The JSR321 reference implementation consists of the following five packages:

- javax.trustedcomputing

  This root package contains only the TrustedComputingException class. Whenever something goes wrong during the usage of the high level API this exception is thrown. It contains all kind of information that caused this exception. If the cause for this exception happens in a lower layer then information about this layer is included as well. This can go down until the TPM is reached. The returned exception messages are designed to be human-readable and they include the TCG compatible error codes.

- javax.trustedcomputing.tpm

  This sub package includes the TPM interface and the TPMContext class. The TPMContext class represents the heart of the JSR321 API and therefore it offers the possibility to connect to a Trusted Platform Module (TPM). Further all kind of TPM depending objects are created through this class. The TPM interface reflects the hardware TPM and its basic functionality. Further this interface allows the usage of the TPM random number generator, enables access to the Platform Configuration Registers (PCRs) and allows to query the TPM itself against various things like its status or its version for example.

- javax.trustedcomputing.tpm.keys

  The keys sub package includes the KeyManager class which is responsible for the TPM based key management. This means that all the different keys which are defined through the various key interfaces in this package can be created, modified and deleted through this class.

- javax.trustedcomputing.tpm.structures

  This package implements all kinds of structures that are needed through the different parts of this implementation. It includes the Digest, PCREvent, PCRInfo, Secret and ValidationData classes.

- javax.trustedcomputing.tools

  The package tools contains three tools for binding user data to a platform state, for sealing data and for signing user data or files with the help of the TPM. The classes are therefore appropriately named Binder, Sealer and Signer.

After the presentation of the three main components it is time to combine these parts to a create our new grid/cloud computing framework.

4.5 A Trustworthy Grid/Cloud Computing Solution

4.5.1 Overview

As discussed above GridGain is one of the most powerful grid/cloud computing frameworks for Java. But without proper security features it is not trustworthy at all in public and open networks. This means that we have no control about the nodes which simply join the network neither about what these nodes are doing with our workunits or data at all. Therefore GridGain should be extended with an authentication and authorization feature to achieve at least the security that only authenticated and authorized users can participate in our grid/cloud computing environment. With a good PKI behind Permis, which will be designed in the next chapter, our framework will already reach a high security state and therefore it will be much more trustworthy than the shown grid/cloud solutions in chapter 3. But with including TC into our framework during the PKI design process much more trustworthiness can be included and used within our computational network.

4.5.2 Needed Software Applications

As mentioned before through this Chapter we need GridGain, Permis and the JSR321 to build our new grid/cloud computing framework. For GridGain the latest stable version from the GridGain homepage [27] should be downloaded and not an experimental version from the SVN.

From the Permis web-site only an outdated version can be downloaded. Therefore the SVN should be used to download the latest version.

The latest TC reference implementation based on the JSR321 specification can be downloaded from the JSR321 development site [35]. A detailed how to for the whole process can be found here [34].

4.5.3 Needed Adaptions to GridGain

Some adaptations to GridGain are needed. By default GridGain uses the TCP communication protocol and multicast to find other nodes. These kinds of settings can be changed at any point of time directly by reconfiguring GridGain, but to let GridGain work together with Permis we have to change one of the Topology SPI implementations. Of course a new one can be added as well to hold still all original GridGain options available. This change is needed to include the feature of an authentication/authorization process. We will add this process then directly in the chosen Topology SPI implementation to achieve our goal that we can verify for every new node if it is really allowed to join the network or not. The alternative would be to try to anchor this process somewhere else in the code of GridGain. So this is done by the reason that the Topology SPI implementation is always called when a new node arrives or when a new workunit should be sent to another node. We can use these two circumstances perfectly to ensure that only nodes can join which can be authenticated and which are authorized to join. But at first we will only ensure that all workunits will only be sent to nodes that are authorized to be in the network.

Our preferred implementation will be for now the “Node Attributes SPI” which represents an included GridGain implementation of the Topology SPI. We chose it because it will need the fewest modifications to include our feature to ask Permis if the current node is authorized to be the network. The idea is to modify the getTopology() method within the “Node Attributes SPI” implementation. This method is called whenever a new task has to be deployed and delivered to some free nodes in the current network topology. With proper modifications this function can be reconstructed to add only nodes which are holding the proper attributes, a authentication key, which is extracted from the attributes and sent to Permis. The PDP from Permis verifies then the authentication key for its validity. If Permis is able to authenticate the node and to verify its authorization to be into the network then the node is added into the network topology for task deployment. Otherwise it is ignored. Further a helper class should be added

which is responsible for all communication tasks between GridGain and Permis. An example adaptation can be found in the appendix chapter A.1.5.

4.5.4 GridGain Client Adaptations

To achieve the goals that we are always aware of what client platforms are in our network and that we always can ensure that new client platforms fulfill our requirement that they are in a thrustworthy state, we have to protect them against modifications. A thrustworthy state means in our case that the client platform is executing exactly our delivered software system without any modifications. The protection mechanisms we will use are indirect, which means we will only be able to detect modifications not to prevent them. Therefore we will combine our modified GridGain client-software directly together with the acTvSM system.

The client-software must be designed in a way that at first the acTvSM platform boots the system into a secure and trustworthy state. After that the signature of the GridGain client-software is verified. Only if the signature of the client-software can be verified, acTvSM allows to unbinding and starting it. The unbinding of the client-software is only possible it the secure and thrustworthy state of the acTvSM platform has really successfully reached. If a different state has reached the unbind operation will in any case fail because of the reason that the system state represents an essential part of the encryption process. Furthermore if the secure state can not be reached because of some modifications then the acTvSM platform will automatically hold the system during the boot process. This is a feature of the current version. Such a modification could be changes in the BIOS or in the platform hardware. After the successful unbinding and start of the client-software the client will automatically connect to the server and request workunits. During the connection process a remote attestation query again the correct system state of the client's software platform. The complete protocol is described in chapter 5.

All further modifications which can be derived trough the PKI can be found in chapter 5.1 to 5.3. Further a schematic view about the registration process, the client joining process and the workunit request can be found there.

4.5.5 What has changed

GridGain has now still all its powerful features but the weakness that it simply adds all found nodes can now be solved by using our new topology SPI implementation instead. Any kind of key can now be created for any client and Permis will automatically verify if this key is valid or not. Based on that decision new nodes are added to our grid/cloud computing network or not. It depends now on a great part on the used and underlying PKI model how secure and trustworthy our new environment becomes. In the final approach in the next Section 5.2 "A PKI for Java based Grid/Cloud Computing" a well suited PKI will be developed and by the help of Trusted Computing a much higher security and trust level can be reached for our framework.

A first example with this simple grid/cloud computing framework can be found in the appendix. Because of the early stage of the framework, the missing PKI features and the comprehensive changes that will follow through chapter 6 only a simple demonstration in this first example is given how it can be achieved to let GridGain and Permis work together.

Chapter 5

A Public Key Infrastructure for Java based Grid/Cloud Computing

5.1 Conceptual Overview

As we have mentioned before, we will include trusted computing features into our new grid/cloud computing framework. These features are based on a chain of trust which again relies on asymmetric cryptography. Therefore we will need a proper PKI to handle the used keys and certificates in a proper way. Moreover we have to design use cases for the user and client platform registration process, the client platform joining process and how the distribution of the workunits should work. For these reasons we have to deal with the questions what are the needed elements of our PKI, what framework modifications do we need for our new grid/cloud computing framework to include this PKI and how can we design the just mentioned three use-cases?

Therefore this chapter centers around the theoretic creation of an appropriate Public Key Infrastructure for our new grid/cloud computing framework. Initially a description about all used components like actors, keys, needed software components and needed framework modifications will be described accurately. The Section “Actors” contain all users which are needed for our PKI and our new grid/cloud computing framework to be usable at all. Section “Keys Used” shows the used keys in detail and which functions they fulfill. “User Signatures” do the same for the used sigantures. The needed hardware components will then be described in the next Section. To utilize our new PKI the Section “Needed Software Components” informs about the needed software components. The last Section before the use-cases will follow is dedicated to the needed framework modifications for our new grid/cloud computing framework.

After that description a detailed model description for all usual use cases will be explained. These model descriptions are containing the three main use cases registration of a new person and its client platform, the adding of a client platform into our network and the process of distributing workunits. Finally an analysis about solvable risks with the help of our framework and these PKI modifications will show what kinds of risks have been successfully eliminated.

5.2 Component Description

5.2.1 Actors

- Developer(-Group)

  At least one developer who maintains the whole system is needed. Further it requires at least one administrator which is responsible for the creation and maintenance of the policy files. The devel-

opers themselves are further responsible for the creation of proper workunits and the registration process.

- Users

  At least one user is needed who is fully registered. This means that the user must have proceeded the complete registration process (like shown in figure 5.1).

- Platform Owner (TPM)

  The platform owner is in charge of the TPM initialisation and usage. Normally it is the client-user itself or the administrator of the client-platform.

5.2.2 Keys Used

- An asymmetric TPM-based AIK Key (server)

  The GridGain server software itself uses the built-in TPM to derive an AIK. This AIK is used to register the server platform at an PrivacyCA server.

- An asymmetric TPM-based AIK key (per client)

  The public part of the AIK is needed during the registration process of the client by the server. Further the private part is used to sign the result of the quoting process during the remote attestation which is done in the network joining process (like shown in figure 5.2). The AIK is further used by the server to verify the quote result signature and by the PrivacyCA [2] to ensure that the client-platform has a real hardware TPM. Therefore the PrivacyCA checks if the AIK has been derived from an Infineon TPM. This is done with the reasoning that the client software should only run on platforms with real hardware TPMs.

5.2.3 Used Signatures

- AIK Certificate

  The AIK Certificate is created during the registration process of the client's platform. It ensures that a real hardware TPM is present on the registered hardware. Currently only Infineon TPMs can be verified because only they offer certificates for all of their TPMs. Further this AIK Certificate is used during every new connection or boot of the client platform by the GridGain client software for the authentication process to show that exactly the registered hardware is trying to connect.

5.2.4 Hardware

- Server

  For running the GridGain server only an appropriate hardware with TPM support is needed. Further this TPM must be activated, the ownership taken and be usable (by the right drivers).

- Client(s)

  Each client needs as well a capable TPM supported hardware. The TPM must be activated, the ownership taken and be usable.

- Network Infrastructure

  For every registered client it must be possible to connect to the server and to hold the connection permanently as long as this client is computing workunits for the server.

5.2.5 Needed Software Components

- acTvSM Platform

  This special Trusted Computing platform is used as the basis for the clients. It enables secure boot and is further used to directly start the GridGain based client software and to be able to perform the Remote Attestation process.

- GridGain - Permis - JSR321 Framework

  Our just before created and only rudimentary framework is used and modified to run this PKI, which is described during this chapter.

- Database

  An encrypted database must be used to hold all registration data in one place accessible. This database is further used to store the revocation list of banned users and platforms.

- PrivacyCA

  The public available PrivacyCA is used to verify during the step one of the registration process that the registering user is really using a platform with a hardware TPM. Currently PrivacyCA allows only to verify Infineon based TPMs.

5.2.6 Mandatory Framework Modifications

- A general integration of TC and therefore TPM support is needed and must be directly added into GridGain to achieve several security solutions. It is needed for example to perform a remote attestation during the connection process of a new client-platform or during the registration process as well. Therefore the JSR321 is used and directly connected with GridGain.

- Further all communication and discovery features of GridGain must be deactivated and only the direct connection of client-platforms to the server should be allowed. Further the collision and failover SPI must be configured to send back a workunit only to the server without asking another node for help. This ensures that classified workunits can only get to the proper clients which are offering the needed security level.

- At least the client software should run directly on the acTvSM platform which enables the feature of a trustworthy and secure boot. During the remote attestation process which happens during the connection process of the GridGain client software the correct boot and the start of the unaltered software must be verified.

- GridGain must be adopted in a way that it allows automatically to transfer from the client nodes the authentication credentials together with the AIK certificate to the server. These two elements are needed during the connection process to verify the identity of the platform hardware and the user. Further the GridGain server must be adopted to be able to receive these credentials and the AIK certificate. A connection from the GridGain server to the PrivacyCA must be made to verify the AIK certificates which must have been created during the registration processes of the client-platforms.

- The AIK certificates of all clients, which have been collected during the registration process of the clients, should be signed by the PrivacyCA and permanently stored. The signing is needed to overcome the risk that someone exchanges a certificate without noticing it. The storage itself is needed to allow revocation and to ensure that the client-platforms are still the registered ones.

- On the server host a logfile must be available where all client connections have been logged. Further all reasons why a client has been banned from the network must be found as well in this logfile.
- An encrypted database must be used to store all registration credentials, the AIK certificates and the AIK public keys from the client-platforms. Further the revocation list must be stored in the database (to be permanently available). This database must contain further a logfile over all deployed workunits, the solver-grid-node and the computation time.
- A differentiation between the workunits should be possible and be available as different security levels.

5.3 PKI Concept Description

This Section shows the three main use cases in which our new PKI is used. This means that all three use cases only contain our new designed behaviour.

5.3.1 Client Registration Process

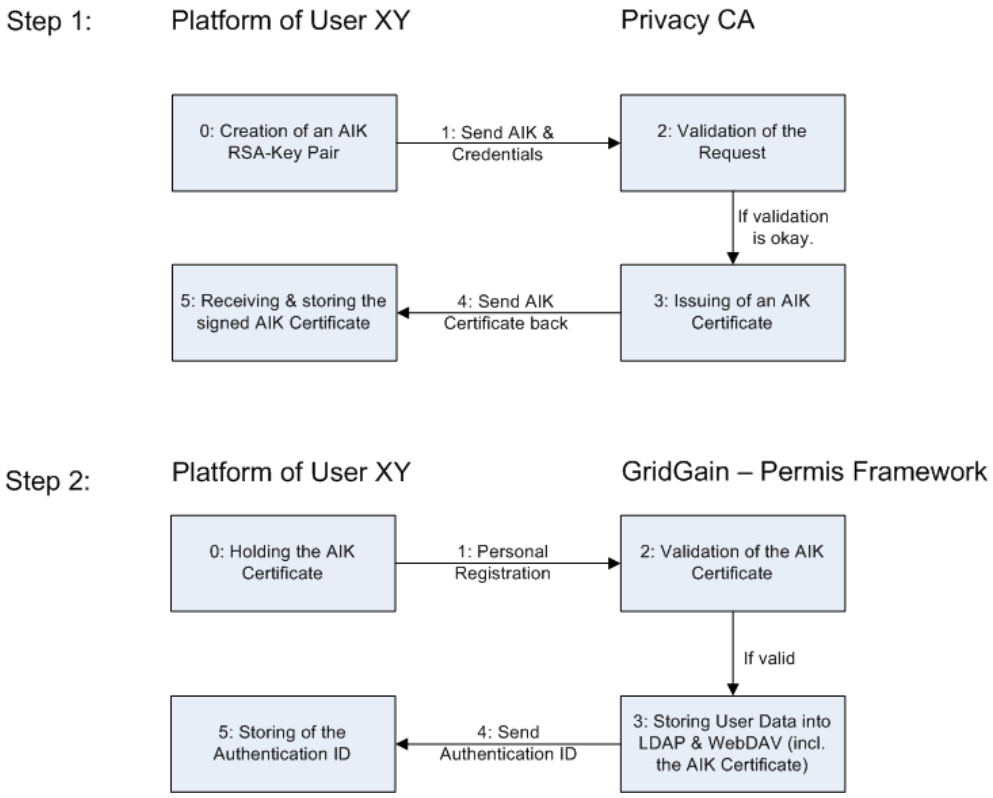


Figure 5.1: PKI Model: Client registration.

For every new user who wants to join the network it is necessary that he owns a TPM-based platform. This TPM must have been activated, the ownership taken and the TPM itself must be usable (with the appropriate drivers). Further an AIK must have been created on this platform. This AIK is needed during the first step of the registration process (like shown in figure 5.1 step 1.0). After the creation of the TPM-based AIK RSA pair, the public part is sent (figure 5.1 step 1.1) to the PrivacyCA server. The PrivacyCA server verifies if the presented AIK has been created on a hardware TPM. Currently the PrivacyCA only supports the verification of Infineon TPMs through their certificates which they are offering. If the validation shows that the AIK is valid (figure 5.1 step 1.2) then an AIK certificate is issued, encrypted with the public received AIK and signed for this client-platform (figure 5.1 step 1.3). This signed AIK certificate is permanently stored at the PrivacyCA to allow a later revocation of this certificate. The AIK certificate is then sent back (figure 5.1 step 1.4) to the client-platform, which receives it, decrypts it and stores it (figure 5.1 step 1.5). The signing is needed to achieve the goal that nobody can easily exchange something without noticing it.

Step 2 of the registration process requires the personal registration of the user. Therefore the user registers himself together with his AIK certificate and his public AIK key (figure 5.1 step 2.1) by the registration authority of the server. The registration authority validates at first the given AIK certificate against the PrivacyCA to check if the certificate is valid and if it has been successfully registered there (figure 5.1 step 2.2). If the AIK certificate is valid then the PolicyEditor is used to verify if the appropriate

role for this user is already existing or not. If not, then it must be created. The next step must be the usage of an registration software utility. This utility is responsible for creating, storing, modifying and deleting of new or already registered users in an existing and encrypted database. During the registration process in step 2 the tool must be capable to create a new user profile which includes the user name, a password, his AIK certificate, his AIK key and his credentials for the correct Permis authorization. The user name and the password is used for the authentication process. The AIK certificate is stored to allow a direct revocation, if needed. Further a proper security level for this client is determined. The new introduced security levels are defined as the letters A, B and C and they are representing the public, internal use only and confidential levels. The level A or confidential is therefore the highest level and C or public represents the lowest security level. This level decides therefore which kind of workunits should be available for this client (figure 5.1 step 2.3). The authentication credentials are then given or sent to the user or its platform (figure 5.1 step 2.4).

5.3.2 Client Joining Process

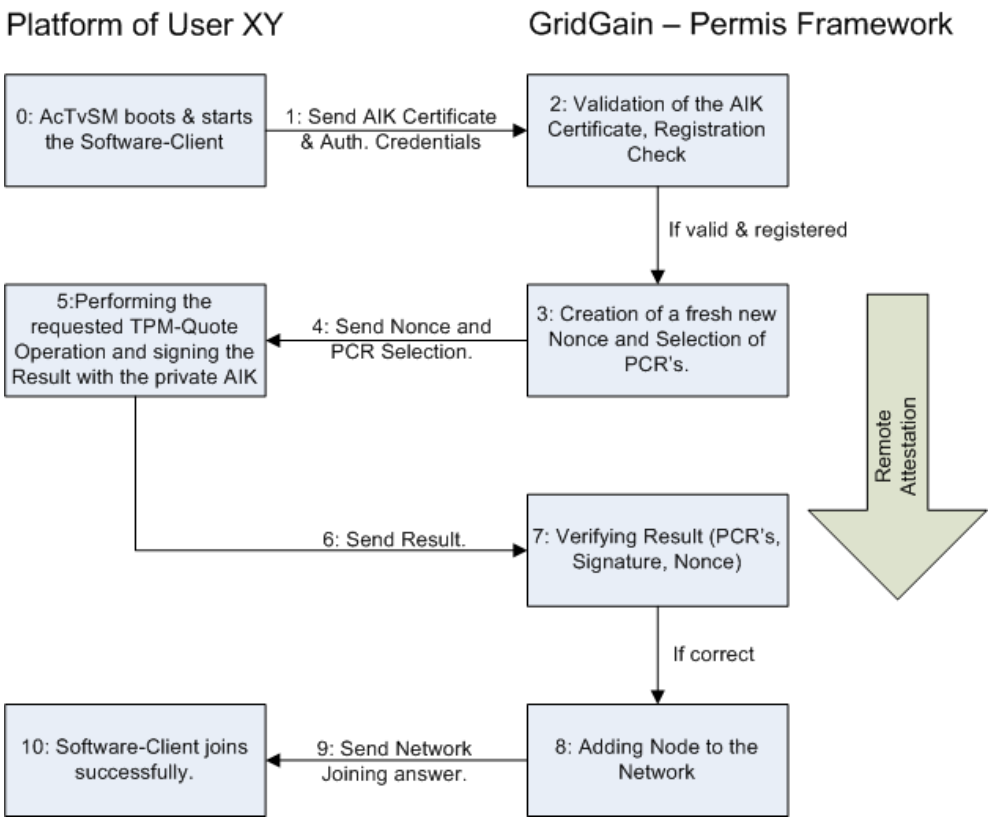


Figure 5.2: PKI Model: A client tries to join the network.

Whenever a user starts his client-computer, it boots at first the acTvSM platform and then the client-software is automatically started (figure 5.2 step 0). After the successful start of the client-software, it tries to connect to the GridGain server to join into the network. His AIK certificate together with his authentication credentials is then sent to the server (figure 5.2 step 1). The server verifies at first the AIK certificate with help of the PrivacyCA to immediately verify if the AIK certificate is valid and already successfully registered (step 2 in figure 5.2).

If the certificate is valid and registered then a new fresh nonce is created and all necessary PCR's are chosen to verify if the correct client-system has been started (step 3 in figure 5.2). The nonce together with the PCR selection list is then sent to the client-platform (step 4 in figure 5.2). The client-platform performs then on the given PCR's and based on the given nonce a TPM-based quote operation (step 5

of figure 5.2). The result of this operation is then finally signed with the private AIK key of the client-platform and then sent back to the server (step 6 of figure 5.2). The Server is now responsible to verify the result which means it validates the PCR values based on the given nonce, the signature and the nonce itself (step 7 in figure 5.2). If everything is correct then the node will be added into the network. If not then the node get marked and is thrown away. If a node tries more than 3 times in a line to connect without success then this node is permanently blocked and the user himself is put on the revocation list and the administrator will be informed about this user and his connection-tryings. This all happens during step 8 in figure 5.2. But if the client is successfully authenticated and authorized then it receives a positive response and is automatically added into the network (step 10 in figure 5.2).

5.3.3 Allocation of Workunits Process

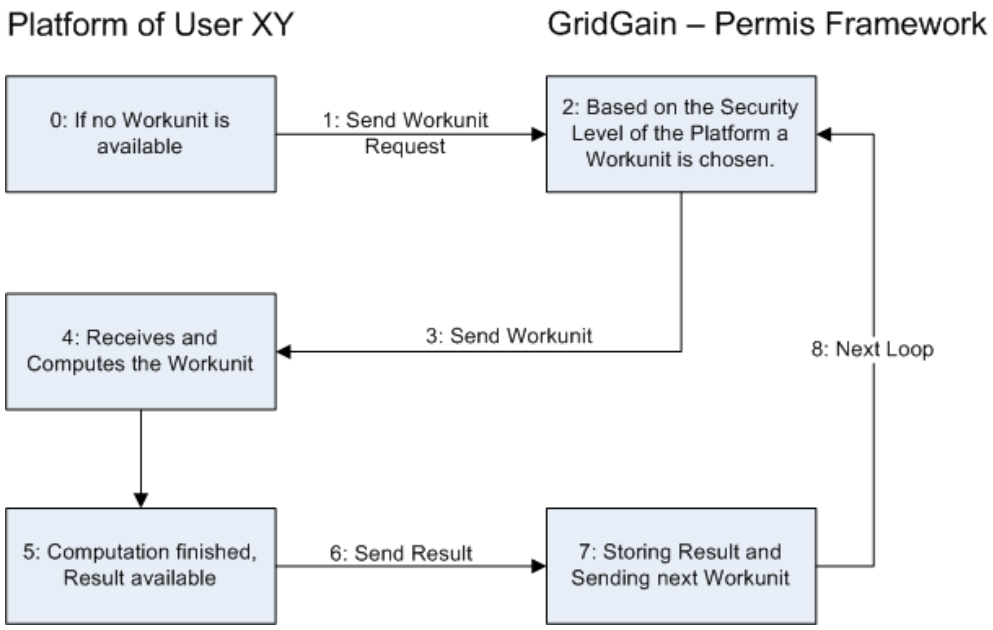


Figure 5.3: PKI Model: Workunit request.

Every successfully registered client tries automatically after the start and whenever no workunit is locally available to connect to the server to retrieve new workunits to solve (step 0 in figure 5.3). Therefore a registered client sends a request for a new workunit to the server (step 1 in figure 5.3). The server verifies always if this client has been joined the network correctly by looking into his node list. Based on the security level of the requesting node an appropriate workunit for this client-platform is chosen (step 2 in figure 5.3). The sent workunit together with the receiving node is logged.

If no workunit with the same security level as the client is available then automatically a lower one is picked up. This security level is set during the registration process and allows the workunit designer to decide which kind of clients should be allowed to compute which workunits. Further it allows sending confidential workunits only to the most trustworthy clients. After the right workunit has been chosen it is automatically sent to the client (step 3 in figure 5.3). The client starts then directly with the computation (step 4 in figure 5.3). As soon as the result is available for this workunit (step 6) it is sent to the server (step 6 in figure 5.3).

The server receives then the result, stores it for further usage and logs the successful receiving of the result for the given workunit (step 7 in figure 5.3). If the IP-address to which the workunit has been sent is different from the IP-address from which the result has been received, then something got wrong and the client-node is kicked out of the network, the user and his platform is set on the revocation list and the administrator is informed by a message in the log file about this incident. This is done because

the connection must be persistent and encrypted and if it breaks then the workunit must be thrown away and a new connection must be established. This is done in this way to know at any point which client is calculating which workunit. If we would allow non-persistent connections we could only guess if the client is still alive and working on the result. If everything is correct then the sever starts again with step 2 to deliver a new workunit to the client-software (step 8 in figure 5.3).

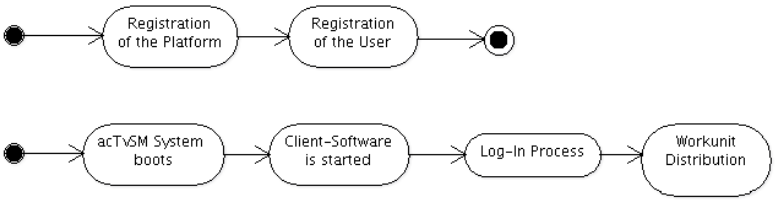


Figure 5.4: PKI Model: The Overall View

Figure 5.4 shows the whole process once again. At first always a registration of the users-platform and then of the user itself is necessary. After a successful registration the system can be booted through the acTvSM system into a trustworthy state. Then the client-software is automatically started and after some tests like remote attestation the client-platform is successfully connected to the server. Only in this final state the download of workunits is possible for the client-software-platforms. The connection between client and server must be persistent. If somehow the connection is lost, then the client rejects the current workunit and tries again to connect to the server and to receive a new workunit.

5.4 Risk Analysis and Management

A lot of different risks exist in grid/cloud computing networks [15][49] and the shown programs and frameworks in chapter 3 are only little or not at all prepared against them. Therefore we will analyze now our new framework in combination with the just before described PKI against our already well-known main security list which is taken from [49].

1. Participants may deliberately report back wrong answers.

   The remote attestation of the acTvSM system is used to verify every booted and started client-software system if it has reached a defined, secure and trustworthy state. If changes have been made to the acTvSM system or the client-software then a different measurement will happen and a connection to the server is no longer possible.

2. Participants may not enforce data integrity or not compute accurately.

   All GridGain clients are started directly with help of the acTvSM system which guarantees a trustworthy and secure booted system. If something gets manipulated during the start process the system would measure it and therefore a unsealing of the client-software would be impossible. Furthermore the client-software performs during its logging process to the server a remote attestation which guarantees the server that the current client-platform has been booted correctly and is at least at the time of the attestation in a stable, trustworthy and secure state. This means in practice that only platforms can join if they are running exactly our software without any changes. Therefore our defined data integrity is guaranteed.

3. Data protection requirements of sensitive data cannot be enforced on remote systems.

   This risk is at least in parts solved by our new framework, because we can guarantee that every client platform is exactly running our software without any changes to the platform or the software itself. Furthermore we can use our new introduced security level system to send sensitive data or workunits only to the most thrustworthy platforms in our network.

4. Theft of confidential data by the participants is possible.

   This risk is not direct solveable by our new framework. But indirectly because we log all communication between the client platforms and our server. Furthermore a security rating exists which gives users (including their client platforms) during the registration process their personal security level. Because of this rating only workunits within their security level or lower get sent to these clients. This additional feature allows dealing with confidential workunits in a way that only the most trustworthy clients receives them to solve. Any existing User is further personally registered and known and therefore it can be made personally responsible for any theft.

5. Theft of intellectual property on the distributed code by the participants is possible.

   This risk is also not directly solveable by our framework but we can reduce it by handling this risk in the same way as the risk before.

Furthermore we have analyzed our new framework against the following risks:

- What happens if the server gets rid of a client which only tries to authenticate with a brute force attack?

  If the server receives from one client more then two wrong authentication keys then this client is automatically marked as a malicious client and is banned out of the network if he was already successfully registered. This means that the IP address of the client-platform and the AIK certificate of this platform is banned. Further the administrator receives a notice about this case through the

log file and the client-platform is blocked for any reconnect to the server with setting him on the revocation list. The username itself is put on the revocation list as well.

- How to deal with the risk that at least one GridGain node is in a way manipulated and reconfigured that it get rid of other nodes and try to steal their jobs for manipulation ?

  Every client-node knows and sees only the server node. It can only get rid of another node if the current used client-node and at least another node got manipulated but because of the reason that a log shows exactly which client-node got which workunits, the server would automatically detect if a result would be sent from another node. The result would then be dropped and both nodes be banned.

- How is the risk solved that not all clients have the same trustworthy state?

  This risk is easily solved by giving the clients during the registration process an individually security level based on their trustworthiness. Every client can then only get workunits within his security level or below. Every workunit is simply marked for which minimal security level it is designed.

- Is it possible that a user exchanges his computing platform without recognizing it ?

  If a user exchanges his platform, then he must reregister his new platform again. If not, then any connect with the new platform will fail and after 3 attempts the user and his platform will be banned and put on the servers revocation list. This happens because the remote attestation will only succeed with the registered TPM and his registered AIK TPM Key.

- How to detect a platform without a hardware based TPM ?

  A user who tries to register with a software based TPM will allways fail during the first registration step. This happens because of the reason that during the registration process the PrivacyCA will try to verify the TPM with an existing certificate from its developer. Currently only Infineon based TPMs are equipped with such an appropriate certificate. Every other used hardware TPM will fail for the moment because of this reason and the software based ones in every case.

Chapter 6

PKI Implementation

6.1 Overview

During this chapter we present very detailed how the before only theoretically described PKI can be implemented. In the first Section, the "Registration Process", we will show the idea of how to use the already existing PrivacyCA tool for the first registration step. Further we present the needed mySQL table settings, including a short description for what they are needed and how the server side registration process can look like. In Section 6.3 "GridGain Modifications" we will present all the changes that must have been done to allow the new framework to work as expected. Therefore we will show a very fine grained solution description, including all changed and new created classes and furthermore how the internal method-call-hierarchy is working. All necessary code changes will only be made in the GridGain code base and therefore the JSR321 and the Permis code will not be modified at all. During the last Section we will describe how the GridServer and how the GridClients are built and started.

6.2 Registration Process

The first part of the PKI must be the implementation of the registration process as shown in Figure 5.1. This is done in two steps. The first step depends on the successful creation of an AIK certificate and then its registration by the server. For this process the existing PrivacyCA implementation [2] can be perfectly used. For the registration process only the client part of the PrivacyCA implementation is needed and therefore it is enough to create a runnable JAR file which automatically starts the client part.

Of course the PrivacyCA client tool can be used directly from the console as well, but it would be nicer to have all libraries, compiled classes and files together in one archive. This simple registration utility can then be used on every client-platform to create the needed AIK certificates and its AIK TPM keyblob.

Per client it is necessary to execute this JAR file. The host and port number should address the available public PrivacyCA test server from the IAIK. This server will be used for all AIK certificate based use cases.

Step two is now to bring these two files to the registration authority of the server. The registration authority is now responsible to verify the given AIK certificate at first against the PrivacyCA to check if the client-platform is really based on a hardware TPM and if he has already successfully registered his hardware. If this precondition is valid then the ACM tool of Permis must be used to create a proper account with the right roles and attributes for this person if the appropriate role doesn't exist already. Further the represented files and the user data must be stored in the encrypted database. For an easy upload of the AIK certificate, the AIK TPM keyblob and the user data a second registration tool has been

developed. This command-line based tool simply takes the 2 files and all user data as arguments, verifies them again and uploads them correctly into the database.

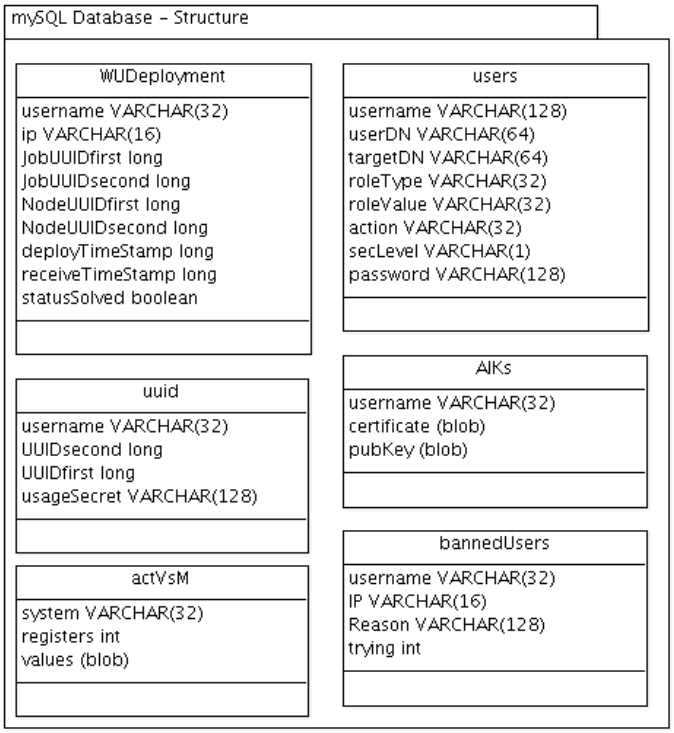


Figure 6.1: mySQL Database - Structure

Therefore an encrypted mySQL database is used on the server. Its five tables are structured as defined in Figure 6.1. The table AIKs holds from all registered users its AIK certificates and its corresponding AIK TPM keyblobs. BannedUsers represents the revocation list for all kinds of banned users and platforms. Users reflects the core table for any registered user. It holds the security level of the registered user, its authorization data for Permis and its authentication password. The table UUID serves simply as a helper table for the AIKs table. It stores the UUID's from the AIK TPM keyblobs and the corresponding secrets to load them. The next table is the WUDeployment. Every workunit that leaves the server is entered into this table by its Job UUID, the UUID from and the IP address of the receiving node. Further a timestamp is set during the deployment and when the result is received. The boolean row statusSolved marks if the workunit has been already solved successfully or not. The last table is the acTvSM table which stores for every GridClient release the appropriate register values.

On the server the UUID from the given AIK TPM keyblob, the AIK certificate, the username and its corresponding password, the security level for the new user and the usage secret for the AIK TPM keyblob are registered. With these arguments the registration tool for the server creates at first a connection to the database. Then it automatically verifies the AIK certificate against the IAIK PublicCA test server. If those two preconditions were successful then it stores at first into the UUID table the username, the UUID from the AIK TPM keyblob and its usage secret. The second step is then the storage of the AIK certificate and its keyblob together with the username into the AIKs table. At last the registration helper tool creates by default a new simple gridworker authorization statement in the users table based on the given security level of the user.

6.3 GridGain Modifications

Figure 6.2 shows GridGain with all modified functions and its corresponding classes. From the original GridGain classes only the GridMulticastDiscoverySPI, the GridAttributsSPI and the GridTaskWorker have been modified. Four new helper classes the JSR321_HelperClass, the PrivacyCA_HelperClass, the Permis_HelperClass and the General_HelperClass have been added to fulfill all needed actions directly centered in the appropriate helper class. The NodeAfterDiscoveryCheck class is used during the authentication/authorization process for any new node.

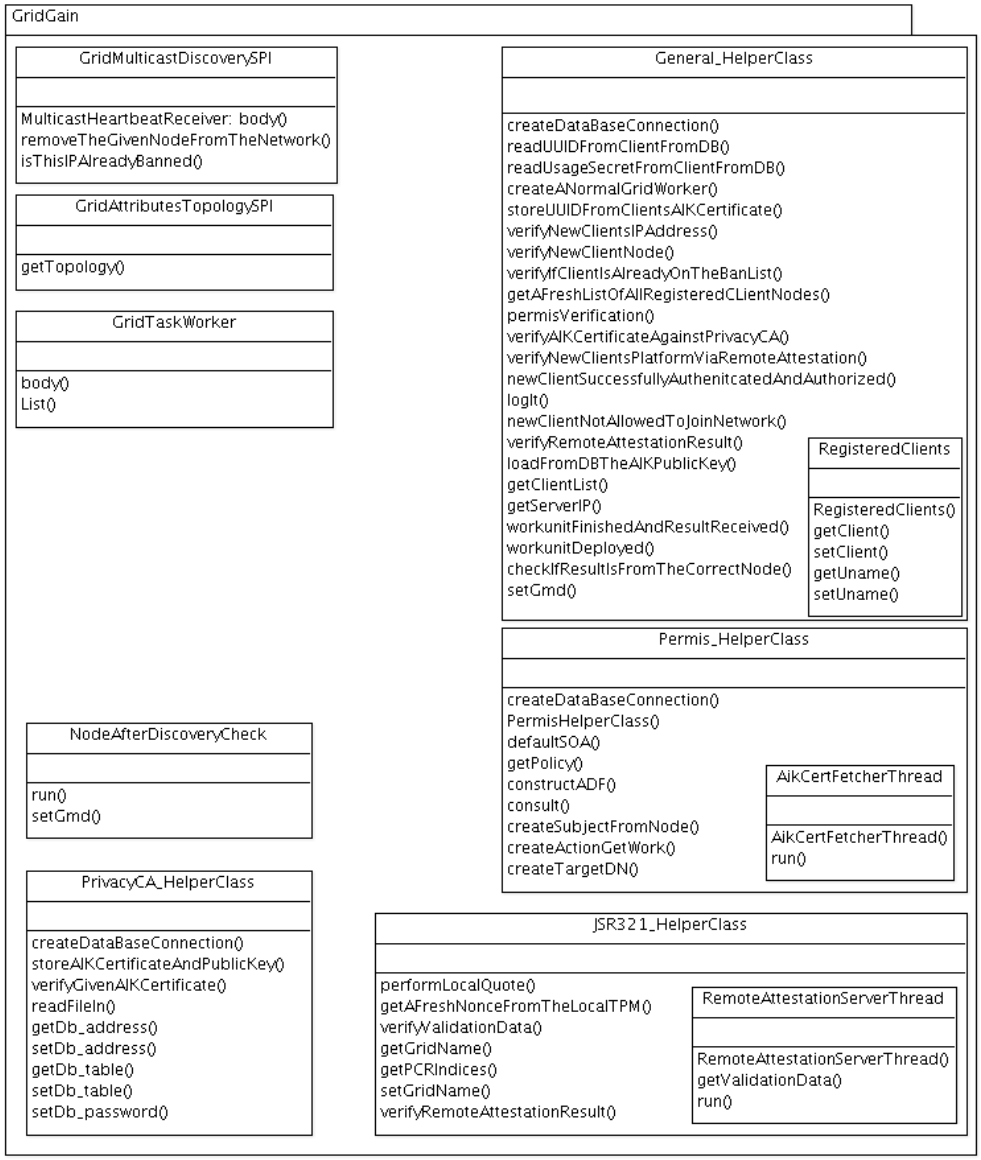


Figure 6.2: GridGain with all its modifications.

6.3.1 GridMulticastDiscoverySPI

The GridMulticastDiscoverySPI class contains the sub class MulticastHeartbeatReceiver. This sub class with its body() function is responsible to process any received heartbeat. Therefore this body function has been modified to act in the following way: For every new node that is found through its heartbeat normally a handshake protocol is started to directly add the new node into the network. The handshake

itself is always started from the node with the higher UUID. This UUID is created during the node start by random and it represents a 128-bit value. The term UUID itself is the short form of universally unique identifier. This behavior has been changed in the way that only the server node is capable to initiate the handshake protocol. Further before such a handshake with a new node is initialized, always an IP address check is performed to verify if this host IP address is not already banned. This is done through the added function isThisIPAlreadyBanned().

If the hosts IP address is not on that list then the server initiates the handshake process and lets the new node indirectly join the network. Indirectly means that the node only joins the network but it will not receive any work-packages until the full authentication and authorization process is finished. For this process a new instance of the NodeAfterDiscoveryCheck class is created and started as a new thread. Only when this instance runs successfully to its end then this new node can finally completely join to the network. If something fails then the node is blocked from the server side which means that it will not receive any workunits or messages again from the server.

In such a case the client can try to restart his client for another try but after three tryings the client will be put on the revocation list and permanently banned from joining the network at all. The server-side log file will protocol all kinds of these events to inform the administrator.

The NodeAfterDiscoveryCheck class has only two methods to offer, the setGmd() and the run() function. After the creation of a new instance the setGmd() function is used to set the actual used GridMulticastDiscoverySPI instance. This instance is needed to inform the GridMulticastDiscoverySPI after all checks with the new client node if it is allowed to join completely or not. The run() function itself calls from the General_HelperClass the verifNewClientNode() function. Depending on its boolean result the new client node is added or banned. If the node must be banned and kicked out of the network then the removeTheGivenNodeFromTheNetwork() function through the GridMulticastDiscoverySPI instance is called.

The verifyNewClientNode() function performs now all necessary checks to verify the new client node. Therefore it calls at first the internal function verifyIfClientIsAlreadyOnTheBanList(). This function serves only as a fast checking and responding call to look if the IP address of the new client node is already banned or not. Therefore this function verifies at first the username and the password of the new client and then it checks if the username or its IP address is on the ban list.

Only if this function detects that everything is all right then the next check is performed by calling the permisVerification() function. This function uses the Permis_HelperClass to load the current Permis Policy, to create based on that policy a new Permis object instance, to load all necessary information about the new client node and its user from the database and then finally to ask the Permis instance if this user is authorized to join the network or not.

If successful then the verifyNewClientNode() function performs automatically the next test by executing the verifyAIKCertificateAgainstPrivacyCA() function. This function is responsible to verify at first the AIK certificate. Therefore a new connection is established to the new client node and its AIK certificate is requested once again. The just received AIK certificate is then at first compared with the one from the database which has been provided during the registration process. Therefore the verifyGivenAIKCertificate() function from the PrivacyCA_HelperClass is called. If those two AIK certificates are identically then the PrivacyCA server is contacted to verify the status of the provided AIK certificate and the signature of the certificate. If the status is still valid then the verifyAIKCertificateAgainstPrivacyCA() reports successful back to the verifyNewClientNode() function.

If all verifications until now were successful then the verifNewClientNode() tries to finally verify the state of the new client node platform through a remote attestation. Therefore an instance of the JSR321_HelperClass is created and the AIK usage secret from the database loaded. Through the subclass RemoteAttestationServerThread a new thread is started which is in charge to perform the whole remote attestation with the client node. This thread opens therefore a new socket connection to the client platform and sends him a just before and fresh created nonce, the desired PCR selection list on which based the

quote must be made and the UUID from the AIK TPM keyblob and its usage secret.

On the client side the JSR321_HelperClass is used to load through the given UUID the identity key into the TPM and then to quote the current system state based on the PCR selection and the submitted random nonce. The result is then encrypted with the AIK private key and sent back to the server.

RemoteAttestationServerThread finally stores now the received results and informs the waiting verifyNewClientsPlatformViaRemoteAttestation() function about the results. The verifyNewClientsPlatformViaRemoteAttestation() function calls now from the JSR321_HelperClass the verifyRemoteAttestationResult() function to verify it the results are valid or not. This function itself verifies at first if the given nonce is the same as the nonce which had been sent back from the client platform. If those two are identical then the stored identity key for this client is loaded into the servers TPM and the internal verifyValidationData() function is called. This function simply calls directly from the JSR321 the validateQuote() function which verifies the signature of the quote and if the result depending on the nonce and the selected PCR selection list is valid or not.

If all these tests have been performed with success then the new client is finally fully added into the network and the newClientSuccessfullyAuthenitcatedAndAuthorized() function is called. This function simply adds the new client to an internal list of all successful registered clients. This list is further used through every workunit deployment to verify quickly if the requesting client node has really joined the network correctly or not.

If at least one of the tests has failed then the internal method newClientNotAllowedToJoinNetwork() is called. At first this function always stores the joining failures into the log file and into the database in the table bannedUsers. This is done because any client is only allowed to have 3 attempts in one line to join into the network. If that amount is reached or crossed then no further try is allowed for this user and his platforms. The plural means that automatically all platforms from the same user are banned if one of his platforms has a missleading behavior.

6.3.2 GridAttributesTopologySPI

The GridAttributesTopologySPI contains the getTopology() function. This function is always called whenever a new workunit needs to be deployed. If no special argument or a whole list of them is given through the initialization of this SPI implementation then it simply adds all existing nodes. Such an attribute could be for example that at least 4 CPU's must be available and/or 2GByte of RAM. Then only nodes would be added that could fulfill these criteria.

For our new framework this behavior has been changed to the following one: Whenever the getTopology() function is now called it verifies at first the security level of the next task. After that, all successful and complete joined clients of the current network gets scanned for their security level. If the security level is A (confidential) then only clients with the security level A can join this new topology for this workunit. If the level is B (internal use) then all clients of the security level A and B can join and if the level is C (public) then every available client is joined into the topology as long as every of these clients have completely and successfully joined the network as described before.

The determination of the security level of the available workunits is done by name extension. Every workunit must end with "_secLevelA", "_secLevelB" or "_secLevelC". This is needed to recognize the security level of the workunit. GridGain itself supports a wide range of annotations. These annotations have to be in the line before a method or class name follows. The annotation "@Gridify" in front of a function lets GridGain automatically deploy this function to any other available node. At first the idea was to annotate the workunits exactly in the same way with an additional annotation code, but in the way of simplicity and clarity it has shown that this kind of annotations are not accessible through the deployment process itself. Only heavy changes of GridGain itself would make this goal functional. Therefore these simple extensions are used right now, because they are allowing a very fast and easy recognizing of the given security level without changing too much.

6.3.3 GridTaskWorker

The last modified class of GridGain is the GridTaskWorker class. The modifications inside this class are only very lightweighted, because only logging functionality has been added. At the end of the body() function the workunitDeployed() function from the General_HelperClass is called. This function simply logs directly into the database in the WUDeployment table all deployed tasks.

At the end of the List() function, which is automatically called whenever a result from a client node returns, the workunitFinishedAndResultReceived() function from the General_HelperClass is called. This function updates the before done entry in the database with the receiving timestamp of the result and further it marks the deployed task as successfully finished, if it had.

6.4 JSR321 and Permis modifications

The JSR321 and the Permis packages are completely unmodified. Only the described changes to Permis to make it compilable under Java 1.5 or higher have been made. All function calls to Permis and the JSR321 are made by the Permis_HelperClass and the JSR321_HelperClass which are located directly into the GridGain package.

6.5 GridServer and GridClient

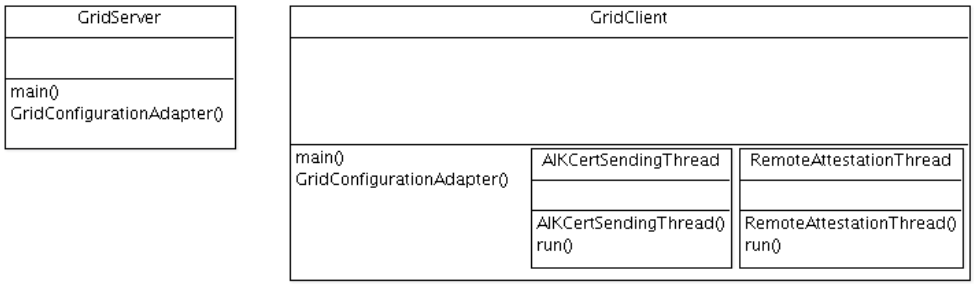


Figure 6.3: The GridServer and GridClient class diagram.

6.5.1 The GridServer

The GridServer is responsible to verify every new available platform against the described PKI model in chapter 5. Further it performs the workunit deployment, its logging and the logging of all kind of malicious actions that happens through malicious client-platforms. The GridServer itself uses the earlier described modifications within the GridGain framework and therefore no further modifications are needed.

Only a new class is needed to start the server-side node out of our new framework. Before the server can be used a GridConfigurationAdapter() method must be created and called in this new class. This method is responsible for the right configuration. In the current version of the framework all original GridGain configuration options are still available. Therefore some of them can be as well activated if necessary. The GridConfigurationAdapter() method adds at first two attributes, the ”username“ ”gridserver“ and the attribute ”server“ with the value ”true“ to the fresh created GridConfigurationAdapter object. These two attributes are needed through the different functions of the helper classes. Next an instance of the GridMulticastDiscoverySPI is created and configured to use the correct multicast address. Further a reference from this instance is set into the NodeAfterDiscoveryCheck and General_HelperClass as a static variable to achieve the possibility to access several needed methods of this instance.

At last an instance of the GridAttributesTopologySpi is created and set into the new created GridConfigurationAdapter. This ensures that the manipulated getTopology() function is always called when a new workunit is deployed.

Now this GridConfigurationAdapter can be used to start our GridServer node by simply calling the static function GridFactory.start(...). For the workunit creation and automatic deployment a simple while(true) thread is currently used in the main() function which deploys every 3 seconds a workunit with a random security level and different input parameters. This test simply demonstrates that only authorized client nodes can receive the appropriate workunits and through the database that every workunit gets successfully solved. Further shows the server-side log file all kind of malicious events and new successfully connected client nodes.

6.5.2 The GridClient

The GridClient represents the computing nodes in our Grid/Cloud computing framework. On every registered client-platform (as described in chapter 5) a GridClient package must run on top of an acTvSM system and their only purpose is to connect to the GridServer and to compute its workunits.

All needed methods to achieve the functionality (as described in chapter 5) are completely available through the already done modifications to the GridGain framework. The GridClient needs similar to the GridServer a proper configuration. This is done by creating a new instance of the GridConfigurationAdapter. Further three arguments will be needed to start the GridClient. These three arguments must be the path of the AIK certificate file, the to be used username and its corresponding password. The idea behind using these three arguments as command line arguments is the easily exchangeability on every client platform.

Into the GridConfigurationAdapter instance the username, the password, the path of the AIK certificate and the current GridClient software version is set as user attributes. Further an instance of the GridMulticastDiscoverySPI is created, configured to the correct multicast IP address and set into the GridConfigurationAdapter. The GridAttributesTopologySPI is as well created and set. At least an instance of the GridNeverFailoverSpi must be created and configured to prevent any node from job stealing. Further it enables the wished feature that if a node can't solve a workunit for various reasons that it always will send the workunit back to the server and not to any other grid client.

After the creation of the correct configured GridConfigurationAdapter the GridClient can be started with GridFactory.start(...) and further two more threads must be separately started: the AIKCertSendingThread and the RemoteAttestationThread. The first one is responsible to wait for the server until it requests from the new GridClient his AIK certificate. Then this thread simply follows the internal protocol and sends the before provided AIK certificate to the server. If everything is all right with the certificate then the username and the password of this GridClient is verified by the server and at the end of the verification process a remote attestation call is made.

Therefore the RemoteAttestationThread is running and waiting for the server. This thread is receiving all necessary information to proceed a local quote of the system state and sends then the result back to the server. If this last check satisfies the server, then the GridClient will finally be able to join the network.

6.5.3 The acTvSM system

The last step before our system becomes fully usable is to integrate it with the acTvSM system. Therefore we have at first to create with qemu [13] a new raw image. On this raw image we install a small and fast gentoo environment. Only the base system with Java and TPM support will be enough. At last our full framework with the GridClient jar file and the registration helper tool for the AIK certificate creation must be copied into this image. Further the GridGain_HOME variable must be set permanently. After these preparations the acTvSM system itself must be installed on every target platform and the just

created image be copied to those platforms. After the successful start of the acTvSM system the image must be sealed to the current PCR configuration. This is needed to protect our GridClient software and to allow the unsealing on other acTvSM platforms only if they have the same unchanged acTvSM system running.

The next step must be the usage of an alternative storage to create and store the AIK certificate and its corresponding AIK TPM keyblob. This is needed to allow still the sealing of the GridClient environment. The registration helper tool must be used to create with help of the PrivacyCA server the needed AIK certificate and its keyblob.

For every GridClient software release, the individual PCR register values must be noticed and stored in the encrypted database. These values together with the system versions are needed during the remote attestation process on the server to determine if the new GridClient is running an unmodified system or not. Further an automatic start of the GridClient software must be available within the image before the system can be sealed.

Chapter 7

Experiments

7.1 Example Grid/Cloud Workflow

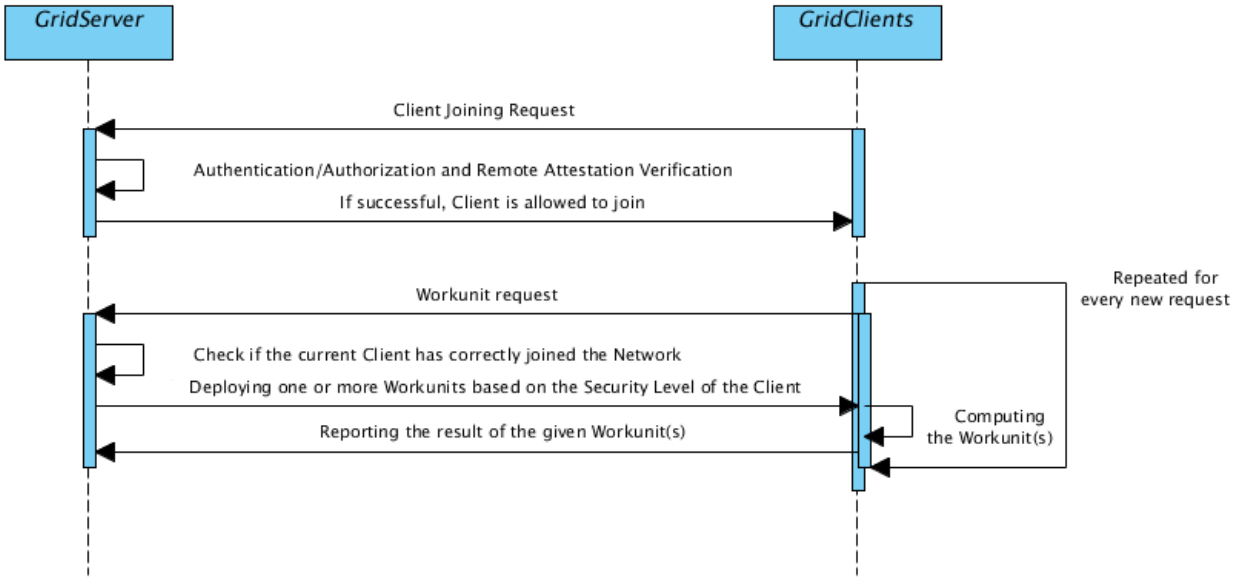


Figure 7.1: The Grid/Cloud Workflow Server/Clients

This chapter centers around the hardware and software environment which has been used during our experiments. Figure 7.1 shows our example Grid/Cloud Workflow between the server and the clients. It consists of one GridServer and three GridClients.

A client is able to handle two actions: to register itself to the GridServer (first action in Figure 7.1) and if successful to request workunits based on their security level (second action in Figure 7.1). The security level is determined during the registration process by the GridServer administration. These two actions are the only allowed ones for the GridClients. The GridServer itself is responsible for the authentication/authorization process (including the remote attestation) for any new GridClient which tries to join the network.

7.2 Annotation of Workunits

The declaration of a workunit is simply done by adding the annotation "@Gridify" in front of a function. The annotation of a function for simply printing "Hello World, I'm running on the GridClient with the IP-Address: xxx.xxx.xxx.xxx" would look like this:

```
1  @Gridify
2  public static void say(){
3    System.out.println("Hello World, I'm running on the GridClient with
       the IP-Address: " +
4    GridFactory.getGrid().getLocalNode().getPhysicalAddress());
5  }
```

A classification for the workunits based on the three existing security levels (A, B, C) is simply realized by marking the whole class of the function with the wished security level. This is done by adding "_secLevelX" at the end of the name of the class. The variable X in "_secLevelX" must be exchanged through the wished security level A, B or C. The simple example from before with security level C would look like this:

```
1  public class justDoIt_secLevelC {
2    @Gridify
3    public static void say(){
4      System.out.println("Hello World, I'm running on the GridClient with
         the IP-Address: " +
5      GridFactory.getGrid().getLocalNode().getPhysicalAddress());
6    }
7  }
```

For a more complicated workunit, like a prime factor determination function, it is possible to annotate the wished function including with its helper class. The role of the helper class is to contain a function to split a complex workunit into smaller and easier computable parts. These parts can then be sent to different clients for a faster computation. Further it must contain a function to reassembly all partial results into the final result. (Parts of this code example are taken from the GridGain website [27].)

```
1  public final class GridPrimeChecker_secLevelC {
2      @Gridify(taskClass = GridifyPrimeTask_secLevelC.class)
3      public static Long checkPrime(long val, long minRage, long maxRange
         ) {
4    // Loop through all divisors in the range and check if the value
       passed
5    // in is divisible by any of these divisors.
6    // Note that we also check for thread interruption which may happen
7    // if the job was cancelled from the grid task.
8    for (long divisor = minRage; divisor <= maxRange
9          && Thread.currentThread().isInterrupted() == false; divisor++)
             {
10       if (divisor != 1 && divisor != val && val % divisor == 0) {
11     return divisor;
12       }
13   }
14   return null;
15     }
16 }
```

The annotation "@Gridify" contains now further the taskClass GridifyPrimeTask_secLevelC.class which is responsible for the correct task splitting. The task splitting is needed to allow multiple GridClients to compute on the same workunit and to collect and rebuild the correct result out of the received partial results. The above shown function includes further a check to verify before any new loop cycle if one of the other computing GridClients has already found a divisor and if yes, then the local computation must be stopped.

The GridifyPrimeTask_secLevelC.class must therefore contain the following two functions:

```
1 ...
2   protected Collection<? extends GridJob> split(int gridSize,
        GridifyArgument arg) {
3   ...
4   }
5
6   public GridJobResultPolicy result(GridJobResult result, List<
        GridJobResult> received) throws GridException {
7   ...
8   }
9 ...
```

These two functions are responsible for the correct task splitting and the recombination of the results from the different GridClients to the complete result.

7.3 Setup of Experiments

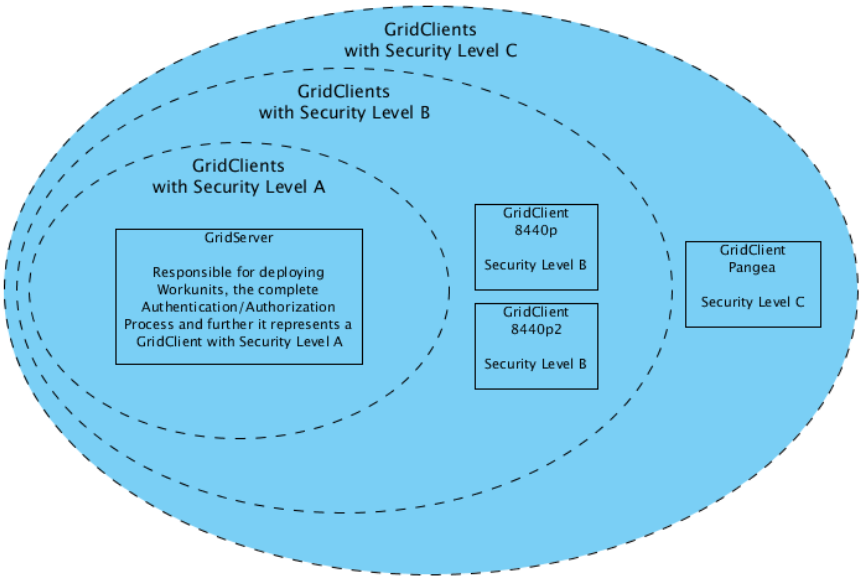


Figure 7.2: Setup of Experiments - The GridClients (with Security Levels)

The GridServer itself is running on a Linux based Gentoo system with a KDE 3.5 window-manager. The used Java environment on the server was based on the Java SE version 1.6.0_20. For the whole development process Eclipse 3.5.1 was used. All tests with the GridServer were directly done by running

it through the Eclipse framework environment. The used MySQL server is a 64bit edition in the version 14.12 Distrib 5.0.90 and the used Permis version is 5.1.1.

The GridServer and the GridClients are using jTSS in the version 0.5.2beta and the JSR321 reference implementation SVN revision 139. The GridGain framework is based on the official stable version 2.1.1.

The clients itself are using the AcTvSM platform based on the public sourceforge version from the October 2010. For the GridClients a Gentoo system based on the minimal installation system from the 09.28.2010 is used. All installed system packages are based on the portage list from the 11.04.2010 and are listed as the latest stable version. The used Java SE version is the 1.6.0_22.

The network topology for the GridClients and the GridServer is based on multicast. For the connection itself the local student network is used.

Figure 7.2 shows the used hardware configuration for the experiments. The GridServer is used for the authentication/authorization process (including the remote attestation process) for the clients. Further it is responsible for the workunit distribution. In our scenario only the server is operating in the security level A, top secret. Within the security level B, company internal, the two given HP 8440p notebooks can be found and for the last security level C, public, only the platform "Pangea" (based on a HP dc9700 platform) is used explicitly.

The test-cases itself are splitted into three Groups according to the three existing security levels to verify that only clients with the appropriate security level are receiving these workunits. Figure 7.2 shows further that GridClients within the security level A can compute workunits for the security levels B and C and that GridClients in the security level B are members of the security level C as well. For the verification of the remote attestation process of the server against the clients some manual tests have been made. These tests have proven that at least an extending of one of the three used and checked PCR registers (16,18,19) results in a fail of this platform during the remote attestation process.

7.4 Performance

A performance check of our GridGain/Permis/JSR321 framework has shown that only the Client-Joining-Process differs extremely against the original GridGain framework. In the original version it takes between 0.1 to 3.1 seconds before a new client is available to the server. This time span is based on the used heartbeat delay of 3 seconds, which means that every 3 seconds a notification to all known nodes is sent to notify them that the sending node is still alive and ready. Furthermore this heartbeat is sent continuously trough multicast and it is used by any node on the grid to find other existing nodes. In our framework this time differs between 11 to 14 seconds because of the the various security checks. It starts with the verification if the current client is already banned (600ms). After the Permis verification, which takes about 3.1 seconds, the AIK certificate of the client is verified live against the IAIK PrivacyCA, which takes about 1 second. Finally the remote attestation of the client can be done. This process takes the bigest part of the needed time span with 7 seconds. But in reality this process is only done once for every Client-Joining-Process and therefore it can be disregarded in the overall view.

The difference between the original workunit deployment algorithm and the modified one in our framework differs only in the area of milliseconds ($<$20ms). In contrast to the original GridGain version the security improvement is very significant. Further should be noticed that real workunits would need for their complete computation at least seconds to minutes and therefore these few milliseconds can be disregarded as well.

In the overall view our GridGain/Permis/JSR321 framework is not really noticeable slower than the original GridGain framework. We have seen only during the Client-Joining-Process a significant time difference.

Chapter 8

Open Issues

To enable a reliable and thrustworthy communication channel between the clients and the server a SSL encryption should be introduced. But it should be noticed, that because of some internal circumstances within GridGain itself, that this feature can only be added with extensive changes to the whole GridGain framework and therefore it was out of scope for this thesis.

A smoother realization of the annotation feature for marking the security levels of the workunits would be nice to have as well. At the point where the deployment of workunits takes place in GridGain, no more access to these annotations is possible. That is the reason why in the current version the workunits become marked by a name extension and not directly by an annotation. To add this feature into our framework it would be necessary to do a lot of changes to the original GridGain architecture.

Some other nice to have features would be the usage of the AcTvSM platform for the server to secure it like the clients or to have a complete framework installation package which builds the complete framework out of the box. Some administration tools for the injection of workunits into a running GridServer system would be very useful as well.

Chapter 9

Conclusions

Although Grid Computing is known and used for more than 20 years, all applications in this area are widely or completely insecure against malicious users, theft of shared data or against producing malicious results.

It might be true that in the past such Grid Computing networks have been used only for scientific research where not security but the data throughput was the most important factor.

Since 2007, Cloud Computing has come more and more into popularity and therefore a changeover from only scientific use to commercial use as well has been made. Cloud Computing represents itself as a high level abstraction above Grid Computing - the underlying Grid Computing layer has remained insecure! At least there has been recently some scientific research about these security concerns, the ENISA [15] report for example.

In this thesis we reviewed the Grid and Cloud Computing sector and it became evident that all surveyed applications have massive security problems. BOINC and folding@HOME, two widely known applications from the Grid Computing environment, are communicating completely without any cryptographic support.

Consider the following attack: Someone manipulates his client for uploading wrong results or for gaining more credit points than he should get for a computation. No checks are done by folding@HOME to prevent this. In the case of BOINC at least 3 participants have always to compute the same workunit and only if they send all the same result back then this result becomes valid. However this kind of protection is only a very primitive solution to prevent wrong results and it is a waste of resources.

In relation to Cloud Computing we reviewed in this thesis the Globus Toolkit and GridGain. Both applications present only frameworks and are not directly usable without modifications. These modifications are including the program behavior and its security features. In the case of the Globus Toolkit some security features like SSH or a certificate service can be used out of the box. However the amount of included security features is very low and they are not able to secure a Cloud application in a way which can be recommended. In contrast to the Globus Toolkit, GridGain is completely built in Java which leads to more security regarding memory leaks and other security concerns of the C-programming language. GridGain itself offers no security features at all which makes it completely insecure in public networks.

Furthermore it has been shown that such networks can be made much more secure than they currently are, therefore the idea for an integrated solution for trustworthy Grid and Cloud Computing was born. Trusted Computing based on a secure and reliable acTvSM platform has been chosen to allow security features which are really state of the art. acTvSM offers a complete measured Linux environment which can be perfectly used to ensure that only in an unmodified system state further applications can be started. Trusted Computing offers as well more security features like remote attestation. This feature can be perfectly used to ensure that a remote system is exactly running the expected software. Any change to the system or an application would result in a different measurement.

The feasibility of creating a new and more secure Grid/Cloud Computing framework by combining GridGain (a Grid and Cloud Computing framework), Permis (a complex and modular authorization system) and JSR321 (a Trusted Computing implementation for Java) was demonstrated.

Of course without a proper Public Key Infrastructure, which is responsible for key creation, storage and revocation, behind such a new combined framework we are still far away from a more secure and trustworthy Grid/Cloud Computing framework. Therefore a usable Public Key Infrastructure was designed. Further we analyzed which kind of risks are solvable by our new Grid/Cloud Computing framework and we have seen that the first two risks from our main security list are completely solveable. These two risks were "Participants may deliberately report back answers" and "Participants may not enforce data integrity or not compute accurately". Furthermore we have seen that the third risk "Data protection requirements of sensitive data cannot be enforced on remote systems" is at least solveable in parts. For the last two risks "Theft of intellectual property on the distributed code by the participants is possible" and "Theft of confidential data by the participants is possible" we are completely powerless. Only indirectly by personally knowing each user and its platform we can make them directly responsible for any theft but we are not able to prevent these two risks.

The practical demonstration has shown that nearly all concepts of the theoretically created Public Key Infrastructure can be implemented. The remote attestation feature, which is used during the Client Joining Process, works exactly as described and allows to ensure that every new client-platform has reached exactly our defined system state with its registered client-platform. If a client-user tries to exchange his client-platform without a new registration of his new hardware then the server will automatically ban the user and its platform during his next connect to the server. All Workunits can be marked to guarantee that they become only deployed to client-platforms with the right security level. Any task deployment and result receiving is permanently logged which guarantees that at any point, if a malicous user is detected, all his results can be made invalid. Further this feature can be used to ensure that the received result is always from the node which has received the task.

The difference between the original GridGain framework and our new Grid/Cloud Computing framework is only noticeable in the field of time measurement. While the registration process itself can not be measured due to the fact that physical presence of the client user and his registration credentials is needed, the client-platform joining process can be. It takes nearly 12 seconds before a new started client-platform is completely connected to the server and is able to receive his first workunits. In contrast to the original GridGain version the waiting period became twelve times longer. Usually this joining process is only needed once while the server and the client are able to hold their network connection. The client selection of all available clients before every workunit deployment takes only a few milliseconds and is similar to the client selection period of the original GridGain. Regarding the massive security improvement, that has been introduced through our new framework, these few milliseconds should be seen as entirely insignificant.

This thesis has shown in theory and in practice that it is possible to improve Grid/Cloud Computing networks in the presented way. Further the results show that with the support of Trusted Computing, Grid/Cloud Computing can be brought to a much higher level of security and trustworthiness.

Appendix

.1 Framework installation

This tutorial describes very accurately how to get, to install and to configure Permis and GridGain. Further it shows a way how these two programs can be combined to get a single and more trustworthy grid/cloud-computing framework out of them. This framework is called explicitly a basic framework because of the current simplicity. A lot of improvements will be added through the different chapters of this thesis.

.1.1 Step 0 - System Requirements

The following software components must be at least installed before continuing: Eclipse, Subversion, Java (Version 1.5 or higher), a Linux based operating system. (This guide should be applicable for Windows as well, but all installation steps have been tested only on Linux based systems.)

.1.2 Step 1 - GridGain Installation

The latest official version of GridGain is (October 2010) 2.1.1 and it can be directly downloaded from the GridGain Website:

http://www.gridgain.com/downloads.html

The name of the file is "gridgain-unix.2.1.1.tar.gz". After downloading, it should be extracted to a new directory called "trustedGridFramework" somewhere on the local hard disk. The subversion directory contains of course newer branches of GridGain as well. The latest version in the official subversion directory is 2.2.0 and it can be downloaded from the following address:

SVN https://www.gridgainsystems.com/svn/repos/gridgain.c64/

ATTENTION: Because of the actuality in the subversion directory, there can be a lot of compiling errors in these versions and therefore the recommended version is the stable version 2.1.1.

After GridGain has been extracted to the local hard drive it must be imported into Eclipse as a new Java Project. Therefore Eclipse should be started, the menu "File → New → Java Project" be opened and as project name "GridGain-2.1.1" be chosen. Under "Contents" → "Create project from existing source" should be chosen and then the path of the just downloaded and extracted GridGain directory be entered. It should look like "/path/to/trustedGridFramework/gridgain-2.1.1".

With the "Next" button some additional Java Settings of the GridGain-2.1.1 project can be modified. Eclipse will add all available libraries from the gridgain-2.1.1 directory by itself to the current project. This is fine. With the "Finish" button the whole process can be finalized and Eclipse will build now the whole project.

But if the current stable version of the GridGain-2.1.1 package hasn't changed then a lot of compile errors will occur. Therefore some additional libraries must be added. The fastest and easiest way to get all needed Jar-packages at once is to download the GridGain-2.1.1 branch via the subversion and to copy then the following libraries from the subversion GridGain-2.1.1/libs folder to our GridGain-2.1.1 libs directory. It would be useful to create in the stable GridGain-2.1.1/libs directory a directory called "externalLibs" in which the following libraries must be:

The whole "wls" directory (including com.bea.core.utils_1.0.0.0.jar, weblogic.jar, wls-api.jar and com.bea.core.weblogic.workmanager_1.0.0.0.jar), the com.ibm.ws.runtime_6.1.0.jar, the whole "jBoss" directory (including jboss-aop-jdk5-4.0.4.jar, jboss-aspect-library-jdk50-4.0.4.jar, jboss-common-4.0.4.jar, jboss-j2ee-4.0.4.jar, jboss-jmx-4.0.4.jar, jboss-system-4.0.4.jar, jbossha-4.0.4.jar, jbossmq-4.0.4.jar), from the "glassfish" directory the appserv-rt.jar, from the "servlet-api" folder the servlet-api-2.4.jar, the whole "coherence" directory (including coherence-3.3.1.jar and tangosol-3.3.1.jar), the "gigaspaces" directory (including gs-lib.jar, gs-space-framework.jar, jsk-lib.jar, jsk-platform.jar, jspaces-6.0.jar, openspaces.jar and reggie.jar), the "mule" directory (including backport-util-concurrent-3.0.jar, commons-beanutils-1.7.0.jar, commons-collections-3.2.jar, commons-digester-1.7.jar, commons-discovery-0.2.jar, commons-io-1.2.jar, commons-lang-2.2.jar, commons-pool-1.3.jar, geronimo-j2ee-connector_1.5_spec-1.0.1.jar, geronimo-jta_1.0.1B_spec-1.0.1.jar, jug-2.0.0-asl.jar and mule-1.3.3-embedded.jar), the jxinsight-core.jar package and the ant-1.6.5.jar package.

All the libraries from the "externalLibs" directory must be added to the GridGain-2.1.1 project. Right-click on GridGain-2.1.1 and then "→ Properties → Java Build Path → Libraries → Add External JARs". Select and add now all copied libraries from the "/path/to/trustedGridFramework/gridgain-2.1.1/libs/externalLibs" folder.

In the last step the following folder and package must be deleted from the project: "examples/jboss-cache" and "src/java/org.gridgain.grid.spi.discovery.mule2"

On a system based on Java 1.5 the project should compile now without any errors. In the case that a Java version higher than 1.5 is used, GridGain will not compile without doing some little fixes. There are two classes that must be adapted. The first one is the "GridExecutorService.java" class from the package "org.gridgain.grid.kernal.executor". It is necessary to change the signatures of the following methods like shown below from

```
1  public class GridExecutorService implements ExecutorService {
2
3     ...
4
5     public <T> List<Future<T>> invokeAll(Collection<Callable<T>> tasks)
          throws InterruptedException {
6       return invokeAll(tasks, 0, TimeUnit.MILLISECONDS);
7     }
8
9     public <T> List<Future<T>> invokeAll(Collection<Callable<T>> tasks,
          long timeout, TimeUnit unit)
10    throws InterruptedException {
11      ...
12    }
13
14    ...
15
16    public <T> T invokeAny(Collection<Callable<T>> tasks) throws
          InterruptedException, ExecutionException {
17      ...
18    }
19
```

```
20    public <T> T invokeAny(Collection<Callable<T>> tasks, long timeout,
          TimeUnit unit)
21    throws InterruptedException, ExecutionException, TimeoutException {
22      ...
23    }
24
25    ...
26
27  }
```

into

```
1   public class GridExecutorService implements ExecutorService {
2
3     ...
4
5      public <T> List<Future<T>> invokeAll(Collection<? extends Callable<T>>
           tasks) throws InterruptedException {
6       return invokeAll(tasks, 0, TimeUnit.MILLISECONDS);
7     }
8
9     public <T> List<Future<T>> invokeAll(Collection<? extends Callable<T>>
          tasks, long timeout, TimeUnit unit)
10    throws InterruptedException {
11      ...
12    }
13
14    ...
15
16    public <T> T invokeAny(Collection<? extends Callable<T>> tasks) throws
          InterruptedException, ExecutionException {
17      ...
18    }
19
20    public <T> T invokeAny(Collection<? extends Callable<T>> tasks, long
          timeout, TimeUnit unit)
21    throws InterruptedException, ExecutionException, TimeoutException {
22      ...
23    }
24
25    ...
26
27  }
```

The second class is called “GridStandardMBean” and can be found in the package “org.gridgain.grid.util.mbean”. There is one function that must be changed from

```
1    public class GridStandardMBean extends StandardMBean {
2
3     ...
4
5     public GridStandardMBean(Object implementation, Class<?> mbeanInterface
          )
6       throws NotCompliantMBeanException {
7       super(implementation,mbeanInterface);
8     }
9
```

```
10     ...
11  }
```

into

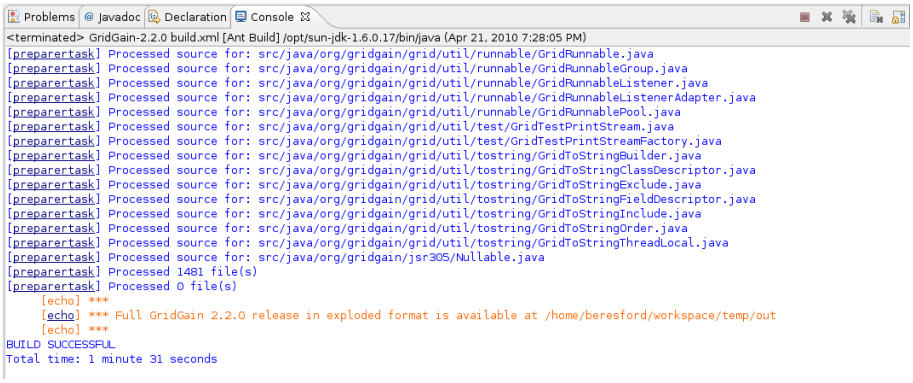
```
1   public class GridStandardMBean extends StandardMBean {
2
3     ...
4
5     public GridStandardMBean(Object implementation, Class mbeanInterface)
6       throws NotCompliantMBeanException {
7       super(implementation,mbeanInterface);
8     }
9
10    ...
11  }
```

Now finally Eclipse should be able to build the GridGain project without any errors.

If instead of the stable version of GridGain (2.1.1) one from the subversion is used then it is maybe desirable that the build.xml file of this GridGain version is working too. Therefore it is necessary to set the “GRIDGAIN_HOME” variable to the root directory of the ”gg-build-2.x.x“ directory. Further a variable called “TMP_DIR” must exist and point to a free directory and finally a variable called “JAVA5_HOME” must exist and point to the Java home directory. This variable must be in any case named “JAVA5_HOME” nevertheless which Java version is used. If problems occur during the setting of these variables into the environment then simply add them directly into the build.xml file. It should look like this:

```
1   <project default=”build.release” name=”GridGain 2.2.0 Main”>
2     <description>
3         GridGain build script.
4     </description>
5
6     <property name=”GRIDGAIN_HOME” value=”/path/to/workspace/gg-base
          -2.2.0/”/>
7     <property name=”TMP_DIR” value=”/path/to/workspace/temp”/>
8     <property name=”JAVA5_HOME” value=”/usr”/>
9
10    <!-- Official and system name of the product. -->
11  ...
```

After these changes the GridGain project should build with the ant-file and within Eclipse without any errors.



GridGain Function test

With this very short “Hello World” program it can be verified if GridGain is working properly. Therefore we must create a new Java Project ”File → New → Java Project“ and call it ”GG-HelloWorld“.

Then a new Class must be added to this new project and named ”GGHelloWorld”. (Right click on the “GG-HelloWorld“ project and select ”New → Class“).

With a ”right-click“ on the ”GG-HelloWorld“ project, continued by selecting the entry ”Properties“ a new window should open and ”Java Build Path“ be selected. On the right side the ”Projects tab“ must be opened and by clicking on ”Add...“ the GridGain-2.1.1 project added.

In the next step the GGHelloWorld.java class must be modified to look like this:

```
1  import org.gridgain.grid.GridException;
2  import org.gridgain.grid.GridFactory;
3
4
5  public class GGHelloWorld {
6
7    public static void main(String[] args) throws GridException {
8
9      GridFactory.start();
10
11     try{
12       say("Hello World");
13     }
14     finally {
15       GridFactory.stop(true);
16     }
17
18   }
19
20   public static void say(String textToSay){
21     System.out.println(textToSay);
22   }
23 }
```

So what have we done ?

The lines 1 and 2 are containing the needed imports for a simple GridGain node to start. Line 9 starts the local GridGain node. This will result in a lot of debug output which shows us that GridGain works properly without any errors. In line 12 a simple method to print out to the console ”Hello World“ is called and finally in line 15 we stop our GridGain node again.

Before this little program can be tested it is necessary to set the GRIDGAIN_HOME variable in Eclipse. If this would be forgotten a ”Default Spring XML configuration file not found“ error would show up. Therefore the following menu must be entered ”Project → Properties → Run/Debug Settings → GGHelloWorld (LaunchConfiguration) Edit → Environment → New”. As “NAME“ must be “GRIDGAIN_HOME” entered and as ”VALUE“ the path to the GridGain home directory (it should look like this: ”/path/to/trustedGridFramework/gridgain-2.1.1” (see figure 1).

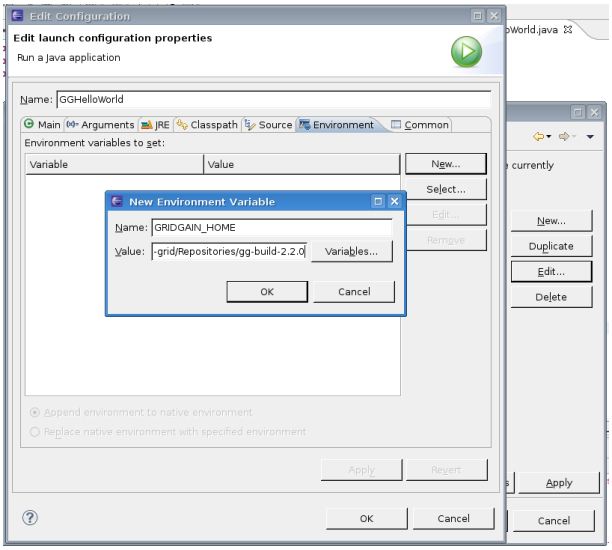


Figure 1: Eclipse - Adding a new environment variable

If this very short test shows no errors at all, then everything until now is working fine. But to check if the just started and stopped node is ready to work together with other nodes it would be necessary to start at least a second node. If a GridGain version from the SVN is used then this test is strongly recommended because from time to time there are versions available which are not willing to work together with other nodes! Therefore a shell must be opened and to the GridGain-2.1.1 home directory be changed. Then the "GRIDGAIN_HOME" variable must be set for the console. This can be done with the command:

export GRIDGAIN_HOME=/path/to/trustedGridFramework/gridgain-2.1.1

If this variable should be set permanently then it should be added into "~/.bashrc".

If the subversion version of GridGain is used then the bin folder must be opened and the "./gridgain.sh" script be called to start a fresh node out of the box. On the stable version it is a little bit more intricately.

A second class must be created in our "GG-HelloWorld" project and it should contain the following code:

```
1  import org.gridgain.grid.GridException;
2  import org.gridgain.grid.GridFactory;
3
4
5  public class SimpleGridNode {
6    public static void main(String[] args) throws GridException,
       InterruptedException {
7
8      GridFactory.start();
9      Thread.sleep(60000);
10     GridFactory.stop(true);
11   }
12 }
```

This simple code only starts a local node, waits 60 seconds and then stops the node. To test GridGain, the second node must be started. It is necessary to set the "GRIDGAIN_HOME" variable for this second class as well if it is started within Eclipse. ("Project → Properties → Run/Debug Settings → GGHelloWorld (LaunchConfiguration) Edit → Environment → New"). The node should start without any errors. The debug output of GridGain should show now 2 nodes with the same amount of available CPU's.

```
>>> ------------------
>>> Discovery Snapshot.
>>> ------------------
>>> Number of nodes: 2
>>> Topology hash: 0x73F8BDEE
>>> Local: 8FB76B09-0FDC-4038-8BA1-43CF7C41B540, 129.27.142.199, Linux amd64 2.6.31-gentoo-r5, beresford, Java(TM) SE Runtime Environment 1.6.0_17-b04
>>> Remote: 44482FE0-379C-427C-82A6-EFF83E42361E, 129.27.142.199, Linux amd64 2.6.31-gentoo-r5, beresford, Java(TM) SE Runtime Environment 1.6.0_17-b04
>>> Total number of CPUs: 2

[17:50:18,776][INFO ][main][GridUpdateNotifier] Your version is up to date.
```

Figure 2: GridGain - 2 Nodes found on the same computer

GridGain is now working fine and as expected.

.1.3 Step 2 - Permis Installation

Currently only version 4.x of Permis is available as direct download from the Permis web site. But it is strongly recommended to use a version newer or equal then 5.0 of Permis. A lot of things are different between those versions and it would make no sense to develop for an outdated version. The latest version of Permis can be directly downloaded from the public subversion directory:

http://projects.cs.kent.ac.uk/projects/svn/trunk/build

To download it a shell must be opened and the current directory changed to the “trustedGridFramework” directory. Then the checkout process can be started with the command:

svn co http://porjects.cs.kent.ac.uk/projects/permis/svn/trunk/build

At last the name should be changed from “build” to “Permis-5.x“

Next should be the adding of Permis-5.x as a new Eclipse project. Therefore Eclipse must be started and the menu ”File → New → Java Project“ be opened. As project name “Permis-5.x” must be chosen. Under “Contents“ → ”Create project from existing source“ must be checked and the path be set to the downloaded ”Permis-5.x“ directory. It should look like ”/path/to/trustedGridFramework/Permis-5.x”.

With the “Next” button the Java settings of the Permis-5.x project can be configured. With the “Finish” button the whole process can be finalized and Eclipse itself will build now the whole project.

Because of compiling errors some (of us) unused parts of Permis must be deleted. First the “data/voms/src” and the “data/gt4/src“ directory, which can be found directly in the Permis-5.x project must be deleted. Next all “issrg.coordination, issrg.gt4, issrg.gt4Plus “ packages (including sub packages) from the Permis-5.x project and the related test packages → ”issrg.test.coordination, issrg.test.gt4, issrg.test.obligation (including sub packages)“ must be removed as well.

The next step must be the removal of all referenced external Jar’s (because of multiple adding of different versions and therefore a lot of compiling errors). With a right click on the ”Permis-5.x“ project and then entering the menu ”Properties → Java Build Path → Libraries“ the libraries section can be opened. Every .jar file must be selected and then removed. Now it is necessary to add exactly the following libraries (with the ”Add External Jar’s“ button) to the project:

”all .jar files which are directly in the folder /path/to/trustedGridFramework/Permis-5.x/lib (except sunxacml-1.2.jar, sunxacml-debug.jar, wsrf_provider_jce.jar“
”all axis2-* files from the folder /path/to/trustedGridFramework/Permis-5.x/lib/Axis2“
”all axiom-* files from the folder /path/to/trustedGridFramework/Permis-5.x/lib/Axis2“
”the bcel-5.1.jar and XmlSchema-1.4.3.jar files from the folder /path/to/trustedGridFramework/Permis-5.x/lib/Axis2“
”the httpcore-4.0.jar, xmlbeans-2.3.0.jar, xml-resolver-1.2.jar and xml-apis-1.3.02.jar from /path/to/trustedGridFramework/Permis-5.x/lib/Axis2“
”all .jar files from /path/to/trustedGridFramework/Permis-5.x/lib/trustpdp“

”the guk.jar and nl.mpi.lookup.jar from /path/to/trustedGridFramework/Permis-5.x/lib/ext“
”all .jar files from /path/to/trustedGridFramework/Permis-5.x/lib/gt4“
”all .jar files from /path/to/trustedGridFramework/Permis-5.x/lib/plugins“
”all .jar files from /path/to/trustedGridFramework/Permis-5.x/lib/gt4+“

Further the packages ”iaik_javax_crypto.jar“ and ”iaik_jce.jar“ must be downloaded from ”http://jce.iaik.tugraz.at“. After the download they must be copied into the lib directory of Permis-5.x and be added as external jar files.

The installation of Permis is now finished and it should build without any errors.

.1.4 Configuration of Permis

Before Permis can be used as an authorization tool some configuration steps must be done. The first step is to download a proper schema, otherwise Permis will only cast null pointer exceptions. A basic ”schema_checking.properties“ file can be downloaded directly from the Permis website:

http://projects.cs.kent.ac.uk/projects/permis/svn/branches/pm/build/data/openpermis/bundle/issrg/pba/rbac/xmlpolicy/schemachecking/schema_checking.properties

Further a proper policy for this schema is needed which can be directly downloaded from:

http://sec.cs.kent.ac.uk/permis/policy50.xsd

After the successful download it is necessary to move the schema_checking.properties and the policy50.xsd file to the following path: ”/path/to/trustedGridFramework/Permis-5.x/src/issrq/pba/rbac/xmlpolicy/schemachecking/”. The last step must be the editing of the schema_checking.properties file. The only line of interest is the line with “schemaLocation=policy50.xsd”. This line must point to the downloaded policy50.xsd file and should look like “schemaLocation=/path/to/trustedGridFramework/Permis-5.x/src/issrq/pba/rbac/xmlpolicy/schemachecking/policy50.xsd”.

Before Permis is ready to use self produced policy files the “TokeType“ class must be modified. This class can be found in the package ”issrg.pba“. This is needed because otherwise only ”UnsupportedOperationException’s“ would be casted from Permis itself during the try to process XML policy files. The following function needs to be changed from:

```
1  public enum TokeType {
2
3    ...
4
5    VOMS_SAML("Voms SAML", "issrg.saml.voms.VOMSSAMLAATokenParser") {
6      public AuthzTokenParser getParser() {
7        return new issrg.saml.voms.VOMSSAMLAATokenParser();
8      }
9    },
10   ...
11 }
```

to

```
1    public enum TokeType {
2
3    ...
4
5    VOMS_SAML("Voms SAML", "issrg.saml.voms.VOMSSAMLAATokenParser") {
```

```
6       public AuthzTokenParser getParser() {
7         return new issrg.shibboleth.ShibbolethAuthzTokenParser();
8       }
9     },
10    ...
11  }
```

Permis is now ready to use.

.1.5 Configuration of GridGain

The GridGain project in Eclipse must be opened and the ”GridAttributesTopologySpi“ class which can be found in the package ”org.gridgain.grid.spi.topology.attributes“ must be modified. In particular a new private variable permisHelper must be created and the ”getTopology(...)” function must be modified.

After the modifications it should look like this:

```
1  ...
2
3  public class GridAttributesTopologySpi extends GridSpiAdapter implements
       GridTopologySpi,
4
5      ...
6
7      /** Named attributes. */
8      private Map<String, Serializable> attrs = null;
9      private PermisHelperClass permisHelper = new PermisHelperClass();
10     ...
11
12     /**
13      * {@inheritDoc}
14      */
15     public Collection<GridNode> getTopology(GridTaskSession ses,
           Collection<GridNode> grid) throws GridSpiException {
16         List<GridNode> top = new ArrayList<GridNode>(grid.size());
17
18           for (GridNode node : grid) {
19
20             try {
21
22               //Load the policy
23               permisHelper.getPolicy();
24               //create a Permis instance
25               permisHelper.constructADF();
26               //save the node files
27         permisHelper.savePermisFilesPerNode(node);
28               //create the question
29               permisHelper.createSubjectFromNode(node.getPhysicalAddress
                     ());
30               permisHelper.createActionGetWork();
31               permisHelper.createTargetDN();
32               //consulting permis
33               String anwser = permisHelper.consult();
34               //check the answer for success, if yes then add the current
                     node to the topology
35               if (anwser.equalsIgnoreCase("0: action succeeded")){
36                 top.add(node);
```

```
37                  }
38
39                  } catch (Exception e) {
40                        System.out.println("error:"+e);
41                    }
42                  if (log.isDebugEnabled() == true) {
43                      log.debug("Included node into topology: " + node);
44                  }
45              }
46          return top;
47      }
48      ...
49  }
```

The getTopology(...) method has now been changed in a way that it only will add nodes to the topology that can be identified by Permis. By taking a closer look into the code the following can be identified:

In line 9 an additional variable called permisHelper which is an instance from the class PermisHelperClass (which will be created during the next step) can be found. In the for-loop from line 18 to 45 any recognized node (locally or from the network and found by GridGain itself) is checked if it fulfills the requirements to be a member of the topology. At first the policy will be loaded (line 23), next (in line 25) a new PermisRBAC object with the just loaded policy file is constructed. Then a method (in line 27) is called which saves the different node attributes to the local hard disk in a proper directory. (All these features can be configured in the PermisHelperClass.) In line 29 the name of the current node is determined, which will be the current ip-address of this node, to create a proper directory for it and the node attributes. From line 30 to 31 a token has been created in dependency to the current node and it's attributes to consult Permis with it. Line 33 finally calls the Permis PDP to check if this node can be authenticated and if it is authorized to get work packages. If the "answer" is "0: action succeeded" then the current node will be added (line 36) to our new topology. All other nodes, which can't be recognized by Permis, will be dropped. The lines 39 to 44 are only for catching exceptions that might happen and for logging, if the log is activated.

Further two new classes must be created in the current package ("org.gridgain.grid.spi.topology.attributes"), which will be called "PermisHelperClass" and "WriteToFileThread".

At first the properties of the GridGain-2.1.1 project must be modified. Therefore a right click on "GridGain-2.1.1 → Properties → Java Build Path → Projects → Add" must be done and the "Permis-5.x" project be selected. This is necessary to let the new to create PermisHelperClass directly talk with Permis.

The PermisHelperClass class should look like this:

```
1  package org.gridgain.grid.spi.topology.attributes;
2
3  import issrg.pba.PbaException;
4  import issrg.pba.Subject;
5  import issrg.pba.TokenType;
6  import issrg.pba.rbac.LDAPDNPrincipal;
7  import issrg.pba.rbac.PermisAction;
8  import issrg.pba.rbac.PermisRBAC;
9  import issrg.pba.rbac.PermisTarget;
10 import issrg.simplePERMIS.SimplePERMISPolicyFinder;
11 import issrg.simplePERMIS.SimplePERMISToken;
12 import issrg.simplePERMIS.SimplePERMISTokenParser;
13 import issrg.utils.RFC2253ParsingException;
14
```

```
15 import java.io.BufferedReader;
16 import java.io.DataInputStream;
17 import java.io.File;
18 import java.io.FileInputStream;
19 import java.io.InputStreamReader;
20 import java.security.Principal;
21 import java.util.Vector;
22
23 import org.gridgain.grid.GridNode;
24
25 public class PermisHelperClass {
26
27   private String permisWorkingDir = "/home/beresford/trusted-grid/
       trustedGridFramework/config/permis/example1-policy/";
28   private SimplePERMISPolicyFinder ssampf = null;
29   private String policy = "policy.xml";
30   private PermisRBAC permisRBAC = null;
31   private String permisFilesToStoreDirectory = permisWorkingDir + "
       GridGainNodes/";
32   private static String file = "filename";
33   private String keyFileName = "/node.txt";
34   // subject variables
35   private String userDN = null;
36   private String targetDN = null;
37   private String roleType = null;
38   private String roleValue = null;
39   private String action = null;
40   private SimplePERMISToken permisToken = null;
41   private PermisAction permisAction = null;
42   private PermisTarget permisTarget = null;
43   private static final String DEFAULT_SOA_DN = "cn=me,o=Amiga,st=Styria,c
       =AT";
44
45   /**
46    * Holds the value of the default policy writer.
47    */
48   private static final Principal DEFAULT_SOA = defaultSOA();
49
50   /**
51    * Computes the default value of the SOA as computed from {@literal
52    * DEFAULT_SOA_DN}. This method catches the exception that may occur.
         If
53    * this method returns null that means that something has gone wrong.
54    *
55    * @return the value of the default SOA
56    */
57   private static Principal defaultSOA() {
58     try {
59       return new LDAPDNPrincipal(DEFAULT_SOA_DN);
60     } catch (RFC2253ParsingException e) {
61       throw new IllegalArgumentException(
62           "Could not compute default policy writer from "
63               + DEFAULT_SOA_DN);
64     }
65   }
66
67   public boolean getPolicy() {
```

```
68      try {
69        File policyFile = new File(permisWorkingDir + policy);
70        ssampf = new SimplePERMISPolicyFinder(policyFile, DEFAULT_SOA);
71      } catch (Exception e) {
72        System.out.println(e.getMessage());
73        ssampf = null;
74      } catch (Throwable th) {
75        ssampf = null;
76      }
77
78      if (ssampf == null)
79        return false;
80      return true;
81    }
82
83    public boolean constructADF() throws PbaException {
84      permisRBAC = new PermisRBAC(ssampf);
85      return true;
86    }
87
88    @SuppressWarnings("unchecked")
89    public String consult() {
90
91      SimplePERMISTokenParser testParserTok = new SimplePERMISTokenParser()
          ;
92      testParserTok.setAuthzTokenParsingRules(ssampf.getParsedPolicy()
93          .getAuthzTokenParsingRules());
94
95      Subject subject = null;
96      try {
97        Vector newCreds = new Vector();
98
99        // create the credentials of principal, i.e., (user <-> roles)
100       // issued by issuer
101
102       newCreds.add(TokenType.SIMPLE_PERMIS_TOKEN.asAttribute(permisToken)
              );
103       // create the subject, i.e. the entity (the roles) recognized by
104       // permis to take the decisions.
105       subject = permisRBAC.getCreds(this.permisToken.getHolderEntry()
106           .getEntryName(), newCreds.toArray());
107
108       // Access control
109       if (!permisRBAC.decision(subject, permisAction, permisTarget, null)
              ) {
110         return "1: the action is not allowed";
111       }
112
113       // here you call the action on the target
114     } catch (PbaException pe) {
115       return "2: invalid input: " + pe.getMessage();
116     } catch (Throwable th) {
117       return "3: run-time error: " + th.getMessage();
118     }
119     return "0: action succeeded";
120   }
121
```

```
122   public void savePermisKeyFilePerNode(GridNode node) {
123
124     String nodeName = node.getPhysicalAddress();
125
126     //delete old keyfile from this node
127     try{
128       File deleteKeyFile = new File(permisFilesToStoreDirectory +
            nodeName + "/node.txt");
129       deleteKeyFile.delete();
130     }catch (Exception nothingToDeleteException){}
131
132     String fileName = node.getAttribute(file);
133     String fileContent = node.getAttribute(fileName);
134
135     File newNodeDir = new File(permisFilesToStoreDirectory + nodeName);
136     //try to create a new directory for node xy
137     newNodeDir.mkdirs();
138
139     fileName = permisFilesToStoreDirectory + nodeName + "/" + fileName;
140
141     WriteToFileThread writeFile = new WriteToFileThread(fileContent,
          fileName);
142     writeFile.start();
143
144   }
145
146   public void createSubjectFromNode(String nodeName) {
147
148     try {
149       FileInputStream fstream = new FileInputStream(
150           permisFilesToStoreDirectory + nodeName + keyFileName);
151       DataInputStream in = new DataInputStream(fstream);
152       BufferedReader br = new BufferedReader(new InputStreamReader(in));
153
154       userDN = br.readLine();
155       targetDN = br.readLine();
156       roleType = br.readLine();
157       roleValue = br.readLine();
158       action = br.readLine();
159
160       permisToken = new SimplePERMISToken(userDN, "cn=me,o=Amiga,st=
              Styria,c=AT", roleType, roleValue);
161
162       in.close();
163     } catch (Exception e) {
164       System.err.println("Error: " + e.getMessage());
165     }
166   }
167
168   public void createActionGetWork() {
169     permisAction = new PermisAction(action);
170   }
171
172   public void createTargetDN() {
173     try {
174       permisTarget = new PermisTarget(targetDN, null);
175     } catch (Exception e) {
```

```
176         e.printStackTrace();
177       }
178     }
179   }
```

The value for the variable called “permisWorkingDir“ needs to be adapted to the newly created “config/permis“ directory and should look like “/path/to/trustedGridFramework/config/permis”. All the further created policies, configurations and more, which are related to Permis, will be saved there. The variable “policy“ calls the name of the currently to use policy which can be found there too. The variable “keyFileName” and ”DEFAULT_SOA_DN” shouldn’t be changed at this moment, because they are quite fine for now.

At last the “WriteToFileThread” class must be created in the same package. It should look like:

```
1  package org.gridgain.grid.spi.topology.attributes;
2
3  import java.io.BufferedInputStream;
4  import java.io.BufferedOutputStream;
5  import java.io.ByteArrayInputStream;
6  import java.io.FileNotFoundException;
7  import java.io.FileOutputStream;
8  import java.io.IOException;
9
10 public class WriteToFileThread extends Thread {
11
12   private String fileName = "";
13   private String content = null;
14
15   public WriteToFileThread(String file, String filename){
16     this.fileName = filename;
17     this.content = file;
18   }
19
20   public WriteToFileThread(){}
21
22   public void run(){
23     try {
24
25       byte[] content_ = content.getBytes();
26         BufferedInputStream in = new BufferedInputStream(new
               ByteArrayInputStream(content_));
27         BufferedOutputStream out;
28
29        out = new BufferedOutputStream(new FileOutputStream(fileName));
30
31         byte[] ioBuf = new byte[4096];
32         int bytesRead;
33         while ((bytesRead = in.read(ioBuf)) != -1) out.write(ioBuf, 0,
               bytesRead);
34         out.close();
35         in.close();
36     } catch (FileNotFoundException e) {
37     } catch (IOException e) {}
38   }
39
40   public void setFileName(String fileName) {
41     this.fileName = fileName;
```

```
42     }
43
44     public void setContent(String content) {
45       this.content = content;
46     }
47   }
```

This class simply creates a new file based on the given file name and fills it with the given String as content.

.2 Example: First steps with the simple GridGain-Permis Framework

The following example shows very detailed how to create a new Permis policy and how to adapt GridGain to use Permis for the authorization process. Because of the missing PKI feature at this point no direct user authentication is done. Further a simple GridGain server and client is described to test the new combined framework.

.2.1 Permis Policy Creation

Our first task should be the creation of a proper Permis policy. Main task of the policy will be to allow all of our virtual company computers the usage of the resource GridGain and further the action on it "getWork".

For the creation of the policy we use the available policy editor for Permis. Therefore we download it directly from the Permis website

http://sec.cs.kent.ac.uk/permis/private/pe/policyEditor_5_1_1.zip.

Normally a password check pops up before the download starts but because we are already registered we know that the login name must be "PERMIS" and the password "letMeIn". We extract the files from our policyEditor_5_1_1.zip file to our Tools directory at "/path/to/trustedGridFramework/Tools".

Of course it would be possible to use the policy editor directly out of our Permis-5.x eclipse project by starting the class "PEApplication" in the package "issrg.Editor2" but it would produce only some error exceptions in the first place. The reason for this are some missing files. If we would copy from our just downloaded policy editor the following files "Banner_Open_Permis.jpg, Banner_Open_Permis2.jpg, Close24.gif, Config24.gif, cross.PNG, empty.xml, gdid.xml, Help24.gif, Help24.JPG, minus.gif, New24.gif, New24.JPG, Open24.gif, Open24.JPG, pe.cfg, pe.jpg, PEApplication_i18.properties, PEApplication_i18_fr _FR.properties, PEApplication_i18_it_IT.properties, PEIFConditions_i18n.properties, permislogo.JPG, plus.gif, Print24.gif, PWApplicaton_i18n.properties, PWApplication_i18n_fr_FR.properties, Save24.gif, tap.xsd, TAPFilesConfig.cfs, tick.PNG, View24.gif, Wizard24.gif, WSDLFilesConfig.cfg, permisPolicy2html.xsl, permisPolicy2html_EN.xsl and permisPolicy2html_FR.xsl" into the directory "/path/to/trustedGridFramework/Permis-5.x/src/issrg/editor2" then the policy editor could be started directly out of Eclipse too.

But for convenience we start our policy editor directly by opening a shell, changing to "/path/to/trustedGridFramework/Tools/policyEditor_5_1_1" and starting it with "java -jar policyEditor.jar".

Two windows should appear and we directly close the Permis welcome window. By the way for some reasons the Permis policy editor seems a little bit unstable from time to time and then only a restart helps. But for creating new policy files it is simply the best available tool.

We select now "New → New Policy" to let the policy editor create a new empty Permis v5 compatible policy. As name we just enter "1.0.0.0" and press the "OK" button (see figure 3).

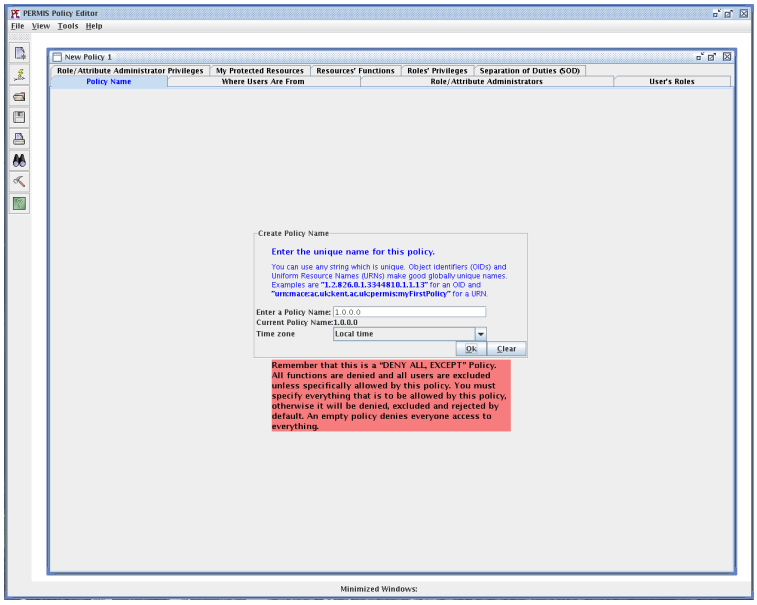


Figure 3: Policy Editor - Create a new Policy

On the next tab “Where Users Are From” we enter as “Nickname:” “companyGrid” and for “Enter LDAP DN” we enter “o=Amiga,st=Styria,c=AT”. The “o=Amiga” stands for organization which is the name of our virtual company, “st=Styria” stands for the state which is Styria in our case and “c=AT” stands for the country which is Austria of course.

(HINT: Permis itself should be compliant to the “RFC 2253“ which means whitespaces shouldn’t have any influences in the policies but fact is that there are still some undetected bugs insight which can cause ”1: Action not succeeded“ error messages from Permis without any hint that a whitespace error caused the violation. Therefore no whitespace should be made for now!)

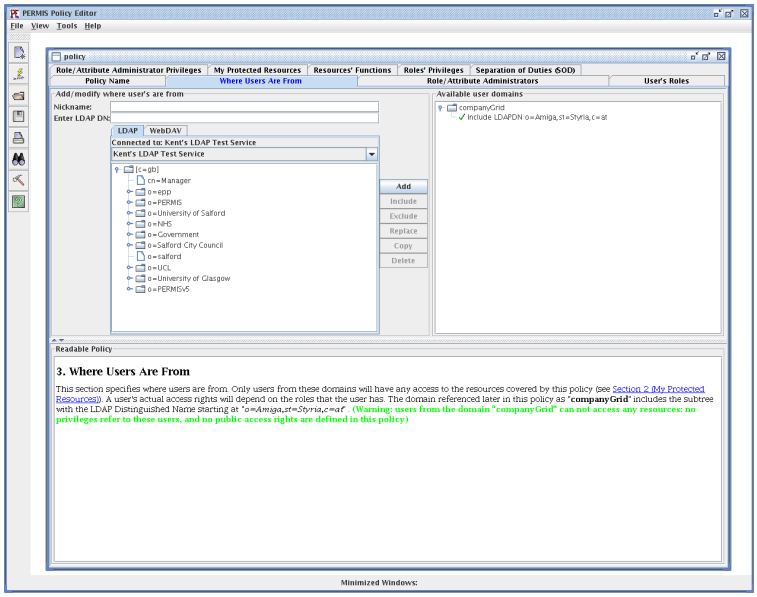


Figure 4: Policy Editor - Adding the user domain

Next we continue with the ”Role/Attribute Administrators“ tab. We have now to add an administrator, therefore we enter as ”Nickname:“ ”administrator“, next we select ”Enter LDAP DN:“ and enter there ”cn=me,o=Amiga,st=Styria,c=AT“.

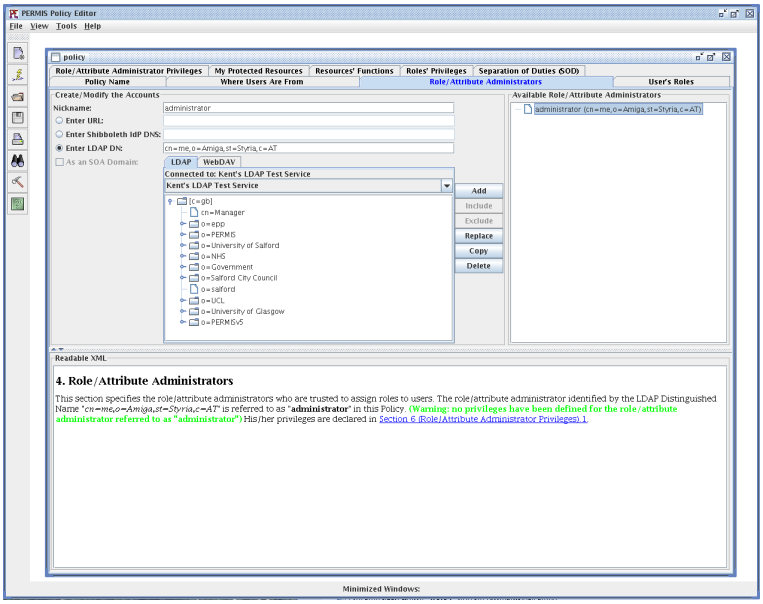


Figure 5: Policy Editor - Adding an policy administrator

Within the tab ”User’s Roles“ we can now define the available ”Roles/Attributes” for this policy, the values that the ”Roles/Attributes” can take and if there is or isn’t any hierarchy between the values. First we select and add “permisRole” from the subitem “Role/Attribute Types”. Next we enter under “Role/Attribute Values” for the “permisRole” the value “gridworker” and then we press the “Add Value” button.

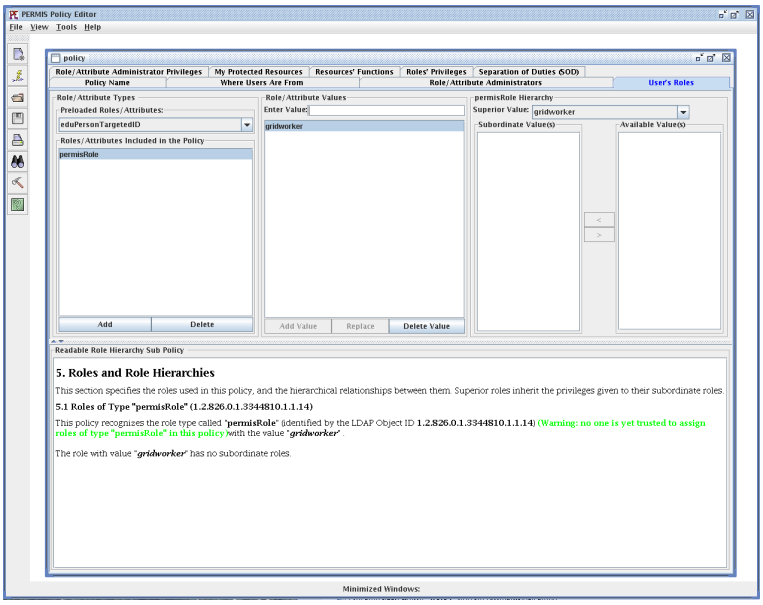


Figure 6: Policy Editor - Adding Roles/Attributes

“Role/Attribute Administrator Privileges” will be our next tab. After clicking on “Add Admin Privileges” a new subitem will be created and opened automatically. In the first third of this subitem we select our “administrator” for “This Role/Attribute Administrator”. In the next third called “Roles/Attributes Name” should be “permisRole” marked and as value “gridworker” be selected. After that we press the “Add” button to accept it. In the last third we have to set from where the users are from. Therefore we select the only possible entry “companyGrid” and at last we deselect the “Delegation” option.

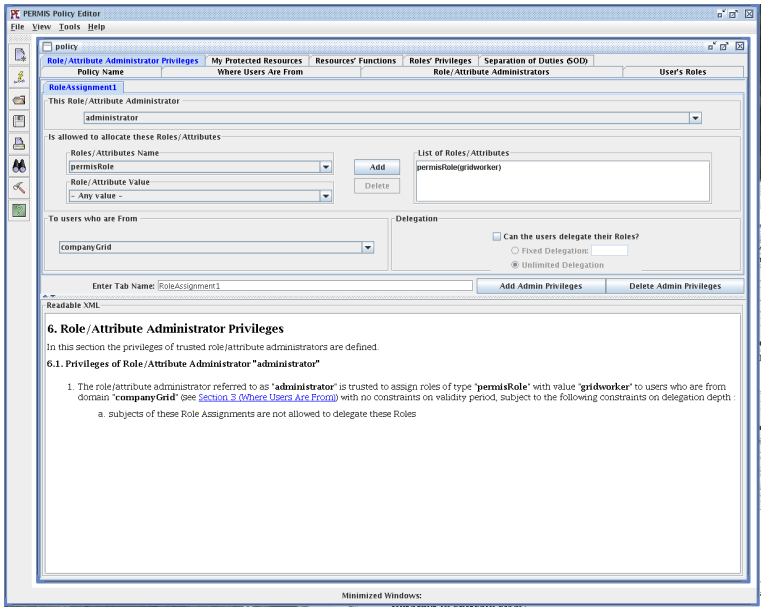


Figure 7: Policy Editor - Role/Attribute Administrator Privileges

Finally we can create a proper entry for our GridGain resource in the next tab “My Protected Resources”. As “Nickname:” we choose “GridGainServer”, then we select “Enter LDAP DN:” and we enter for it “o=Amiga,st=Styria,c=AT”, which means that any authenticated and authorized member of our company from Styria located in Austria can access our GridGain server for work packages.

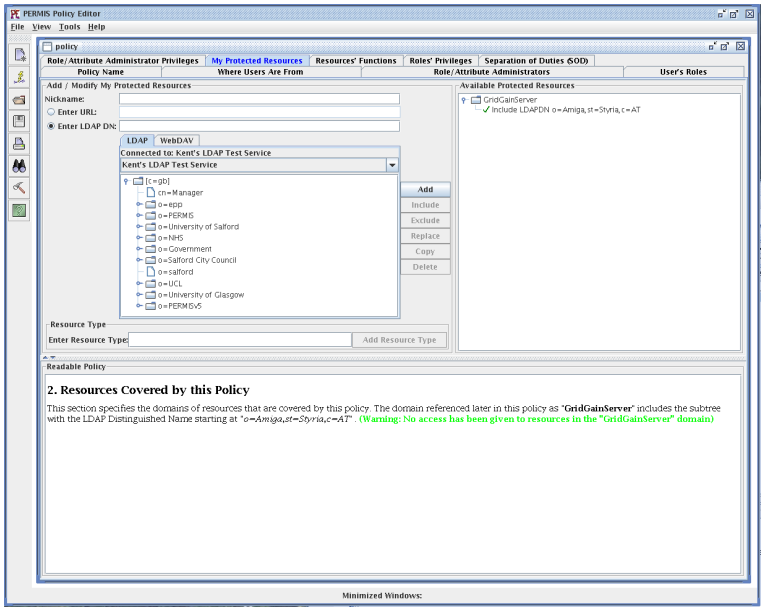


Figure 8: Policy Editor - Adding of a new resource

Under “Resources’ Functions” we have to add the possible functions for our “GridGainServer” resource. We select on the left side our “GridGainServer” under “Can Act On Resource” and then we have to enter at least one “Function Name” for our resource. We call the function “getWork” and accept it with a click on “Add Function”.

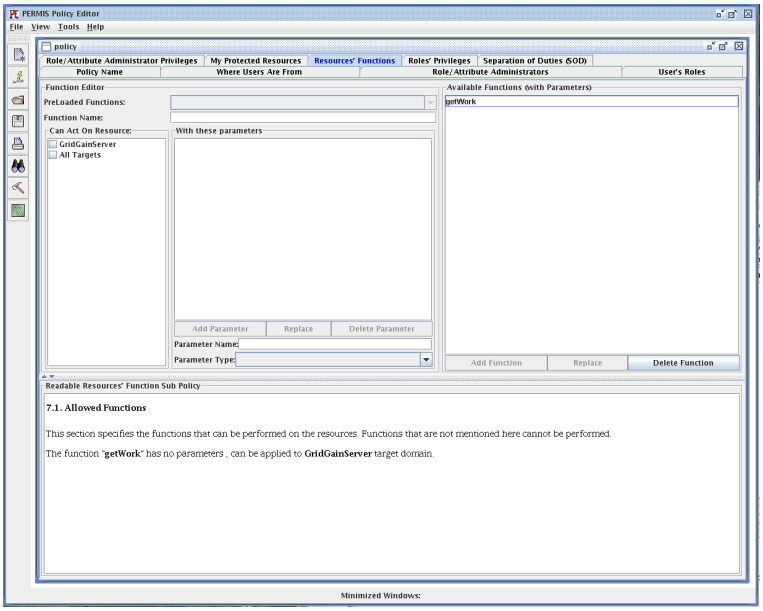


Figure 9: Policy Editor - Adding functions of the new resource

At last we have to enter the “Roles’ Privileges” tab to control who is allowed to access our “GridGainServer“ resource. Therefore we create with ”Add Roles’ Privileges“ a new subitem (called TargetAccess1). In the first third we select ”permisRole“ as ”Roles/Attributes Name“ and ”gridworker“ as value for it. With a click on ”Add“ the selection becomes valid. In the second third (called ”Available Functions“) we select ”getWork“ and accept it again with a click on ”Add“. The last third should hold the value ”GridGainServer“ .

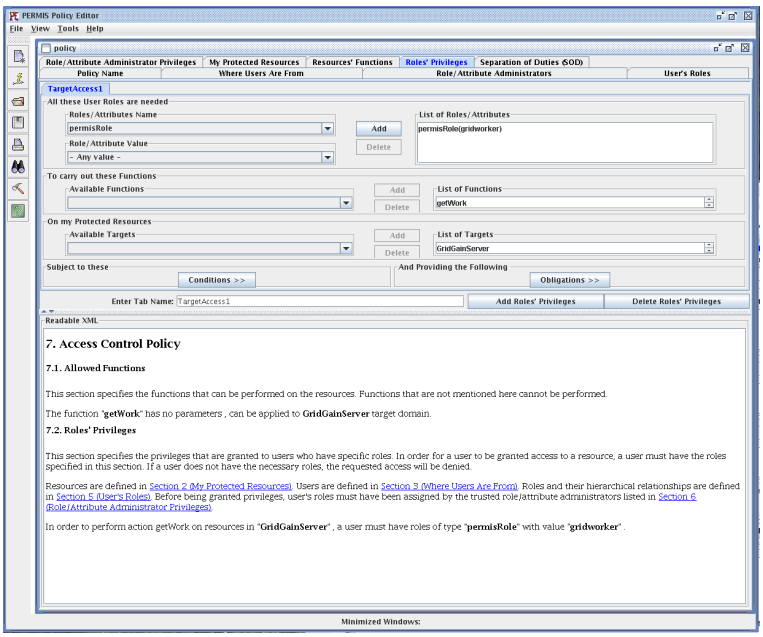


Figure 10: Policy Editor - Configuration of the new resource

Now we can save our new policy under ”/path/to/trustedGridFramework/config/permis/example1-policy/policy.xml“. If no warning occurs during the save operation, which can be done by selecting from the menu ”File $\rightarrow$ Save as“, then the policy is correct. Otherwise the warning will contain hints what kind of errors happened and how they can be removed.

If we open the XML policy with an editor it should look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<X.509_PMI_RBAC_Policy OID="1.0.0.0">
    <SubjectPolicy>
        <SubjectDomainSpec ID="companyGrid">
            <Include LDAPDN="o=Amiga, st=Styria, c=at"/>
        </SubjectDomainSpec>
    </SubjectPolicy>
    <RoleHierarchyPolicy>
        <RoleSpec OID="1.2.826.0.1.3344810.1.1.14" Type="permisRole">
            <SupRole Value="gridworker"/>
        </RoleSpec>
    </RoleHierarchyPolicy>
    <SOAPolicy>
        <SOASpec ID="administrator" LDAPDN="cn=me, o=Amiga, st=Styria, c=AT"
            />
    </SOAPolicy>
    <RoleAssignmentPolicy>
        <RoleAssignment ID="RoleAssignment1">
            <SubjectDomain ID="companyGrid"/>
            <RoleList>
                <Role Type="permisRole" Value="gridworker"/>
            </RoleList>
            <Delegate Depth="0"/>
            <SOA ID="administrator"/>
            <Validity/>
        </RoleAssignment>
    </RoleAssignmentPolicy>
    <TargetPolicy>
        <TargetDomainSpec ID="GridGainServer">
            <Include LDAPDN="o=Amiga, st=Styria, c=AT"/>
        </TargetDomainSpec>
    </TargetPolicy>
    <ActionPolicy>
        <Action ID="getWork" Name="getWork">
            <TargetDomain ID="GridGainServer"/>
        </Action>
    </ActionPolicy>
    <TargetAccessPolicy>
        <TargetAccess ID="TargetAccess1">
            <RoleList>
                <Role Type="permisRole" Value="gridworker"/>
            </RoleList>
            <TargetList>
                <Target>
                    <TargetDomain ID="GridGainServer"/>
                    <AllowedAction ID="getWork"/>
                </Target>
            </TargetList>
        </TargetAccess>
    </TargetAccessPolicy>
</X.509_PMI_RBAC_Policy>
```

.2.2 Adaption of GridGain for Permis

We have already the ”GridAttributesTopologySpi“ class in a way adapted that it only adds GridGain nodes which can be successfully authenticated and authorized through Permis. But what we have missed was that we need individual credentials for any GridGain node so that Permis can make decisions based on that credentials.

For simplicity this credential file will be only a plain text file. A more trustworthy and secure authentication and authorization process will be introduced during Chapter five and six! Every node, including the server itself, should have it's own file which will have to look like this (for the server):

```
CN=server,O=AMIGA,ST=Styria,C=AT
O=AMIGA,ST=Styria,C=AT
permisRole
gridworker
getWork
```

Explanation: The first line contains the name of the user including its user domain which must be defined like shown. The second line contains the target domain, the third one the role type, the next one the role itself and the last one the action that should be done. We should save this simple key file in the following directory ”/path/to/trustedGridFramework/config/permis/“ as “node.txt” on every node.

Now we have all ingredients together to develop our software for the local company clients to join them into our local grid.

.2.3 The Client's software package

The creation of the client's software package is now quite simple. We open Eclipse, create a new Project called GG-HelloWorld and then we create there a new class called “SimpleGridNode”. We have to modify our new and empty class to look like this:

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.Serializable;
import java.util.HashMap;
import java.util.Map;

import org.gridgain.grid.GridConfigurationAdapter;
import org.gridgain.grid.GridException;
import org.gridgain.grid.GridFactory;
import org.gridgain.grid.spi.topology.attributes.
    GridAttributesTopologySpi;


public class SimpleGridNode {

   private static String keyFile = null;

   public static void main(String[] args) throws GridException,
       InterruptedException {

     if (args.length != 1) {

```

```
24        System.out.println("
              ------------------------------------------------------------------------");
25        System.out.println("To Start the node correct write java -jar
              SimpleGridNode.jar /path/to/node.txt");
26        System.out.println("GridNode is now aborting!");
27        System.exit(0);
28      }
29
30      keyFile = args[0];
31      GridConfigurationAdapter gca = addKeyFileToThisNode();
32      GridFactory.start(gca);
33
34      while(true){
35        Thread.yield();
36      }
37    }
38
39    private static GridConfigurationAdapter addKeyFileToThisNode() {
40
41      GridConfigurationAdapter gf = new GridConfigurationAdapter();
42      GridAttributesTopologySpi topSpi = new GridAttributesTopologySpi();
43      Map<String, Serializable> attrs = new HashMap<String, Serializable
            >(20);
44
45      File f = new File(keyFile);
46
47      String fileName = "filename";
48      String fileContent = getStringFileContent(f.getAbsolutePath());
49      attrs.put(fileName, f.getName());
50      attrs.put(f.getName(), fileContent);
51
52      gf.setUserAttributes(attrs);
53      gf.setTopologySpi(topSpi);
54
55      return gf;
56    }
57
58    private static String getStringFileContent(String fileName) {
59      try {
60        BufferedInputStream in = new BufferedInputStream(new
              FileInputStream(fileName));
61        ByteArrayOutputStream bs = new ByteArrayOutputStream();
62        BufferedOutputStream out = new BufferedOutputStream(bs);
63        byte[] ioBuf = new byte[4096];
64        int bytesRead;
65        while ((bytesRead = in.read(ioBuf)) != -1)
66          out.write(ioBuf, 0, bytesRead);
67        out.close();
68        in.close();
69        return bs.toString();
70      } catch (Exception e) {
71      }
72      // @TODO change transformation to a fixed one! Currently it's
            depending
73      // on the system
74      return null;
75    }
```

```
76 }
```

Further we have to add with a right click on “GG-HelloWorld → Properties → Libraries” the “spring-2.5.6.jar” package which can be found at “/path/to/trustedGridFramework/gridgain-2.1.1/libs”. Now it should compile without any error.

What have we done in this class?

The lines from 1 to 13 are only containing the needed imports for our code. At line 20 our main method starts with a short check if the right amount of arguments is present or not (line 22). If not, a short introduction how to start our software will be shown (line 24 to 26) and then the program kills itself in line 27. In line 30 we take the one and only given argument as the path to our credential file. Line 31 creates a new GridConfigurationAdapter object with the help of the addKeyFileToThisNode() method (line 39 to 56). This method creates a new GridConfigurationAdapter and a new GridAttributesTopologySpi object and stores there the given credential file as an attribute into the GridConfigurationAdapter. Further it configures the GridConfigurationAdapter to use the GridAttributesTopologySpi to find and add new nodes. This is necessary because we have currently only the GridAttributesTopologySpi class changed to fulfill our wish that only nodes can join our topology that get positively recognized by Permis. From line 58 to 76 the getStringFileContent(...) helper function can be found which realizes a simple function to read-in a normal text file and return it as a String.

The next step must now be the creation of an executable JAR package which can be rolled out. Therefore we make a right click on “GG-HelloWorld → Export...”. In the opening window we select the subitem “Runnable Jar File” and press the “Next” button. As “Launch configuration” we select “SimpleGridNode - GG-HelloWorld”. As “Export destination:” we enter the name “SimpleGridNode.jar” for our new JAR package and as “Library handling” we select “Copy required libraries into a sub-folder next to the generated JAR”. With a click on the “Finish” button Eclipse starts with the building process of our runnable JAR package.

Now we are ready to deploy our software on any company computer. We can start it by remote over ssh or local in a shell with “java -jar SimpleGridNode.jar /path/to/node.txt”. Only the node.txt file must be adopted for any computer/user.

If the SimpleGridNode has started successfully it should look like this:

```
>>> ------------------
>>> Discovery Snapshot.
>>> ------------------
>>> Number of nodes: 1
>>> Topology hash: 0x29392AC6
>>> Local: 8DAA1558-D141-4AB1-9AD6-7DA69658ED99, 129.27.142.199, Linux amd64 2.6.31-gentoo-r5, beresford, Java(TM) SE Runtime Environment 1.6.0_19-b04
>>> Total number of CPUs: 2

- Update status is not available.
-

>>> ---------------------------------------------
>>> GridGain ver. 2.1.1-26022009 STARTED OK in 3348ms.
>>> ---------------------------------------------
>>> OS name: Linux 2.6.31-gentoo-r5 amd64, 2 CPU(s)
>>> OS user: beresford
>>> VM information: Java(TM) SE Runtime Environment 1.6.0_19-b04 Sun Microsystems Inc. Java HotSpot(TM) 64-Bit Server VM 16.2-b04
>>> VM name: 27443@core2Duo
>>> Optional grid name: null
>>> Local node ID: 8DAA1558-D141-4AB1-9AD6-7DA69658ED99
>>> Local node physical address: 129.27.142.199, eth0
>>> GridGain documentation: http://www.gridgain.org/product.html
```

Figure 11: A simple GridNode (started)

On our GridGain server node it is very important to make the following changes to the “Run Configuration” because otherwise no work package would be deployed. To activate this feature on the server we have to add the following argument “-ea ”-javaagent:/path/to/trustedGridFramework/gridgain-2.1.1/libs/aspectjweaver-1.5.3.jar” ” under “SimpleServer → Run/Debug Settings → SimpleServer → Edit... → Arguments → VM arguments”. Further the aspectj directory must be set. This can be done in the menu “Simple Server → Run/Debug Settings → SimpleServer → Edit... → Classpath → User

Entries (must be selected) → Advanced → Add External Folder". We need to set the following path: "/path/to/trustedGridFramework/gridgain-2.1.1/config/aop/". Now the GridGain server is able to deploy any code to other accepted nodes.

An example GridGain Server that only prints "Hello World" randomly on one of the nodes in the current topology could look like this:

```
1  import java.io.BufferedInputStream;
2  import java.io.BufferedOutputStream;
3  import java.io.ByteArrayOutputStream;
4  import java.io.File;
5  import java.io.FileInputStream;
6  import java.io.Serializable;
7  import java.util.HashMap;
8  import java.util.Map;
9
10 import org.gridgain.grid.GridConfigurationAdapter;
11 import org.gridgain.grid.GridException;
12 import org.gridgain.grid.GridFactory;
13 import org.gridgain.grid.gridify.Gridify;
14 import org.gridgain.grid.spi.topology.attributes.
       GridAttributesTopologySpi;
15
16 public class SimpleServer {
17   /**
18    * @param args
19    * @throws GridException
20    */
21   private static String keyFile = "/home/beresford/trusted-grid/
       trustedGridFramework/config/permis/node.txt";
22
23   public static void main(String[] args) throws GridException {
24
25     GridConfigurationAdapter gca = addKeyFileToThisNode();
26     GridFactory.start(gca);
27
28     //after here start your grid/cloud application
29     try{
30       say("Hello World");
31     }
32     finally {
33       GridFactory.stop(true);
34     }
35   }
36
37   @Gridify
38   public static void say(String textToSay){
39     System.out.println(textToSay);
40   }
41
42   private static GridConfigurationAdapter addKeyFileToThisNode() {
43
44     GridConfigurationAdapter gf = new GridConfigurationAdapter();
45     GridAttributesTopologySpi topSpi = new GridAttributesTopologySpi();
46     Map<String, Serializable> attrs = new HashMap<String, Serializable
         >(20);
47
48     File f = new File(keyFile);
```

```
49
50      String fileName = "filename";
51      String fileContent = getStringFileContent(f.getAbsolutePath());
52      attrs.put(fileName, f.getName());
53      attrs.put(f.getName(), fileContent);
54
55      gf.setUserAttributes(attrs);
56      gf.setTopologySpi(topSpi);
57
58      return gf;
59    }
60
61    private static String getStringFileContent(String fileName) {
62      try {
63        BufferedInputStream in = new BufferedInputStream(new
              FileInputStream(fileName));
64        ByteArrayOutputStream bs = new ByteArrayOutputStream();
65        BufferedOutputStream out = new BufferedOutputStream(bs);
66        byte[] ioBuf = new byte[4096];
67        int bytesRead;
68        while ((bytesRead = in.read(ioBuf)) != -1)
69          out.write(ioBuf, 0, bytesRead);
70        out.close();
71        in.close();
72        return bs.toString();
73      } catch (Exception e) {
74      }
75      // @TODO change transformation to a fixed one! Currently it's
            depending
76      // on the system
77      return null;
78    }
79  }
```

We have now successfully created our executable GridGain package.

Within this example we have now shown how easily a new Permis policy file can be created and how GridGain must be adopted to use Permis for the authorization verification process before any new client node is allowed to join the network. Further we have built a simple GridGain server and client.

List of Figures

Bibliography

[1] Public-Key-Infrastructure.svg. http://upload.wikimedia.org/wikipedia/commons/3/34/Public-Key-Infrastructure.svg (2007) (Cited on pages 8 and 93.)

[2] IAIK/OpenTC PrivacyCA. http://trustedjava.sourceforge.net/index.php?item=pca/apki (2009) (Cited on pages 42 and 51.)

[3] Ati's opencl homepage. http://ati.amd.com/technology/streamcomputing/opencl.html (2010) (Cited on page 21.)

[4] The cern. http://public.web.cern.ch/public/ (2010) (Cited on page 1.)

[5] Jxinsight. http://www.jinspired.com/products/jxinsight/ (2010) (Cited on pages 27 and 34.)

[6] Nvida's cuda homepage. http://www.nvidia.com/object/cuda_home_new.html (2010) (Cited on page 21.)

[7] Seti@home. http://setiathome.ssl.berkeley.edu/ (2010) (Cited on page 1.)

[8] A software architecture for introducing trust in java-based clouds. Berlin, Trust in the Cloud Workshop (2010) (Cited on pages 16 and 93.)

[9] Grid computing - computational grids. http://www.gridgainsystems.com/wiki/display/GG15UG/Grid+Computing (2011) (Cited on page 3.)

[10] Anderson, D.P.: Boinc: A system for public-resource computing and storage. In: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing. pp. 4–10. GRID '04, IEEE Computer Society, Washington, DC, USA (2004), http://dx.doi.org/10.1109/GRID.2004.14 (Cited on page 17.)

[11] Barroso, L., Dean, J., Holzle, U.: Web search for a planet: The google cluster architecture. Micro, IEEE 23(2), 22 – 28 (2003) (Cited on pages 1 and 3.)

[12] Beberg, A.L., Ensign, D.L., Jayachandran, G., Khaliq, S., Pande, V.S.: Folding@home: Lessons from eight years of volunteer distributed computing. Parallel and Distributed Processing Symposium, International 0, 1–8 (2009) (Cited on page 21.)

[13] Bellard, F.: Qemu - the open source processor emulator. http://wiki.qemu.org/Main_Page (2010) (Cited on page 57.)

[14] Caronni, G.: Walking the web of trust. In: Enabling Technologies: Infrastructure for Collaborative Enterprises, 2000. (WET ICE 2000). Proeedings. IEEE 9th International Workshops on. pp. 153 –158 (2000) (Cited on page 10.)

[15] Catteddu, D., Hogben, G.: Cloud computing - benefits, risks and recommendations for information security. European Network and Information Security Agency (2009) (Cited on pages 1, 29, 49 and 65.)

[16] Corporation, F.: Cvcell. http://www.fixstars.com/en/img/EN20071128_OpenCV.pdf (2007) (Cited on page 21.)

[17] Daemen, J., Rijmen, V.: The Design of Rijndael - AES - The Advanced Encryption Standard. No. ISBN 3-540-42580-2, Springer Verlag (2002) (Cited on page 5.)

[18] David Chadwick and Gansen Zhao and Sassa Otenko and Romain Laborde and Linying Su and Tuan Anh Nguyen: PERMIS: A Modular Authorization Infrastructure. www.cs.kent.ac.uk/pubs/2008/2834/content.pdf (2008) (Cited on pages 11 and 35.)

[19] Denker, G., Millen, J., Miyake, Y.: Cross-domain access control via pki. In: Policies for Distributed Systems and Networks, 2002. Proceedings. Third International Workshop on. pp. 202 – 205 (2002) (Cited on page 10.)

[20] F.Ferraiolo, D., Kuhn, D.R., Chandramouli, R.: Role-Based Access Control. No. ISBN 1-58053-370-1, Artech House Publishers (2003) (Cited on page 11.)

[21] Foster, I.: Globus toolkit version 4: Software for service-oriented systems. Journal of Computer Science and Technology 21, 513–520 (2006) (Cited on page 24.)

[22] Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. No. ISBN 1-55860-475-8, Morgan Kaufmann Publishers, Inc, 340 Pine Street, Sixth Floor, San Francisco, CA 94104-3205, USA (1999) (Cited on page 3.)

[23] Gagliardi, F., Jones, B., Reale, M., Burke, S.: European datagrid project: Experiences of deploying a large scale testbed for e-science applications. Lecture Notes in Computer Science, vol. 2459, pp. 255–264. Springer Berlin / Heidelberg (2002) (Cited on page 3.)

[24] Grawrock, D.: Dynamics of a Trusted Platform: A Building Block Approach. No. ISBN: 1934053171 9781934053171, Intel Press, 1st edn. (2009) (Cited on page 14.)

[25] Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography. No. ISBN 0-387-95273-X, Springer Verlag (2004) (Cited on page 6.)

[26] Ivanov, N.: Grid application in 15 minutes. http://www.gridgain.com/screencast/grid_app_in_15min/screencast.html (2010) (Cited on page 28.)

[27] Ivanov, N., Setrakyan, D.: Gridgain. http://www.gridgain.com (2010) (Cited on pages 26, 32, 33, 38, 60 and 93.)

[28] Kaspersky, K., Chang, A.: Remote code exectution through intel cpu bugs. http://nchovy.kr/uploads/3/303/D2T1-KrisKaspersky-RemoteCodeExecutionThroughIntelCPUBugs.pdf (2008) (Cited on page 19.)

[29] Lioy, A., Ramunno, G., Vernizzi, D.: Trusted-computing technologies for the protection of critical information systems. www.mirlabs.org/jias/lioy.pdf (2009) (Cited on page 14.)

[30] Manual K.A. and Karthikeyan P.: Cloud computing - a paradigm shift. Global Journal of Computer Science and Technology 10, 68–72 (2010) (Cited on page 26.)

[31] Mao, W., Jin, H., Martin, A.: Innovations for grid security from trusted computing. http://forge.gridforum.org/sf/go/doc8087 (2005) (Cited on page 29.)

[32] Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Guide to Elliptic Curve Cryptography. No. ISBN 0-8493-8523-7, CRC Press (2001) (Cited on page 6.)

[33] Network Associates: An introduction to cryptography. http://www.cl.cam.ac.uk/~rja14/Papers/SE-05.pdf (1998) (Cited on pages 5, 6 and 93.)

[34] Podesser, S.: Jsr321 - a how to. https://verify.iaik.tugraz.at/research/bin/view/Main/SiegfriedPodesser (2009) (Cited on page 38.)

[35] R. Toegl et al.: JSR 321: Trusted Computing API for Java. Java Comunity Process (2008), http://jcp.org/en/jsr/detail?id=321 (Cited on pages 36 and 38.)

[36] Safford, D., Kravitz, J., Doorn, L.v.: Take control of TCPA. Linux J. 2003, 2– (August 2003) (Cited on page 14.)

[37] SANS Institute InfoSec Reading Room: PGP: A Hybrid Solution. http://www.sans.org/reading_room/whitepapers/vpns/pgp-hybrid-solution_717 (2010) (Cited on page 10.)

[38] Schneider, A.: Saas, paas, iaas - waas? http://blog.doubleslash.de/saas-paas-iaas-%E2%80%93-waas/ (2009) (Cited on pages 4 and 93.)

[39] Strassmann, P.A.: Google - A Model for the Systems Architecture of the Future. http://www.strassmann.com/pubs/gmu/LectureV4.pdf (2005) (Cited on page 3.)

[40] Strongin, G.: Trusted computing using amd 'pacifica' and 'presidio' secure virtual machines technology. Information Security Technical Report, Elsevier, Volume 10, Issue 2, 2005, pp. 120-132 (2009) (Cited on page 14.)

[41] Sugano, H., Miyamoto, R.: Opencv implementation optimized for a cell broadband engine processor. In: Digital Signal Processing Workshop and 5th IEEE Signal Processing Education Workshop, 2009. DSP/SPE 2009. IEEE 13th. pp. 182 –187 (2009) (Cited on page 21.)

[42] Taufer, M., Anderson, D., Cicotti, P., Brooks, C.: Homogeneous redundancy: a technique to ensure integrity of molecular simulation results using public computing. In: Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International. p. 119a (2005) (Cited on page 22.)

[43] The Trusted Computing Group: TCG Specification Architecture Overview. http://www.trustedcomputinggroup.org/files/resource_files/AC652DE1-1D09-3519-ADA026A0C05CFAC2/TCG_1_4_Architecture_Overview.pdf (2007) (Cited on pages 13, 14 and 93.)

[44] Theodorakopoulos, G., Baras, J.: On trust models and trust evaluation metrics for ad hoc networks. Selected Areas in Communications, IEEE Journal on 24(2), 318 – 328 (2006) (Cited on page 10.)

[45] Tögl, R., Winkler, T., Nauman, M., Hong, T.: Towards platform-independent trusted computing. In: STC'09 Proceedings; in CCS 2009 Co-Located Workshops' Compilation Proceedings. pp. 61 – 66. Association of Computing Machinery (2009) (Cited on page 36.)

[46] University, S.: Folding@home. http://folding.stanford.edu (2010) (Cited on page 22.)

[47] University of Chicago: Globus toolkit. http://www.globus.org/toolkit/ (2010) (Cited on pages 24 and 93.)

[48] U.S. Department of Commerce: Data encryption standard (des). http://www.cl.cam.ac.uk/~rja14/Papers/SE-05.pdf (1999) (Cited on page 5.)

[49] Vejda, T., Toegl, R., Pirker, M., Winkler, T.: Towards trust services for language-based virtual machines for grid computing. pp. 48 – 59. TRUST 2008, LNCS 4968 (2008) (Cited on pages 1, 29 and 49.)

[50] Weise, J.: Public Key Infrastructure Overview. http://vlib.eitan.ac.il/digital_signiture/main_pdf/publickey.pdf (2001) (Cited on page 8.)

[51] Yajima, J., Shimoyama, T.: Matrix representation of conditions for the collision attack of sha-1 and its application to the message modification. In: Echizen, I., Kunihiro, N., Sasaki, R. (eds.) Advances in Information and Computer Security, Lecture Notes in Computer Science, vol. 6434, pp. 267–284. Springer Berlin / Heidelberg (2010) (Cited on page 7.)