Agent security in Distributed Control Systems: The case of Tango

Sergi Blanch-Torné[1], Ramiro Moreno Chiral[2], Francesc Sebé Feixa[2]

[1] Escola Politècnica Superior, Universitat de Lleida. Spain.
sblanch@alumnes.udl.es
[2] Departament de Matemàtica. Universitat de Lleida. Spain.
{ramiro,fsebe}@matematica.udl.es

DRAFT
December 2, 2013

Abstract. [3]
Distributed Control Systems is a solution that mixes two concepts: Industrial Control and Distributed Systems. Nowadays this type of solution is widely spread in disseminated resources by networks that can be open accessed or, in the best case, only protected by firewalls. An attack to these systems can be considered very serious, specially when they can be used in energy grids. Too often, during the developments, the security is forgotten in favour of time schedule restrictions.
Tango is an example of this Distributed Control Systems where by its isolated environment inside an institution (firewalled cases) has helped to avoid the security issues. But this is a defect that blocks this systems to arrive to a new horizon of industries. Recently voices have raised the problem and a request has arrived to introduce this goal to the system core.
As in the past has happen with the world wide protocols, like web, e-mail and others, it is a requirement for exposed to attacks systems to have security protocols to cover the security over the different levels of the communication.
Keywords: Distributed Control Systems, Cryptography[4] Engineering, Critical Information Infrastructure Protection (CIIP)

1 Introduction

Tango is a software tool supported by the *Consortium*[5] where 8 of the member institutions are synchrotrons[6] and the ninth is a neutron source[7]. Apart from this institutions there are other projects that uses Tango to control the research facility and recently the industry have shown interest to use it in their processes. But this industry have pointed a very important feature not yet implemented in this system: the security. And not concerning only the classical IT environmental security, that is a institution/user choice, the feature is about the use of cryptology to mathematically protect the system.

A goal of ensure Tango must produce an outcome as similar as the *TLS* is for the web navigation or e-mail, but not only. The final goal to provide security must be involve the other levels of the Open System Interconnection model (the OSI layers) by provide end-to-end agent authentication and encryption. Event than a migration environment should allow to coexist with non secured access, the setup of the system must provide a complete transparent security. Concepts like the certificates shall be integrated to not increase the current complexity of a Tango installation.

The instances running of Tango are in a very specialized situation, but the scientific facilities may not be so different than industry applications. Tango runs over different architectures and

[3] Partially supported by grants MTM2010-21580-C02-01 (Spanish Ministerio de Ciencia e Innovación), 2009SGR-442 (Generalitat de Catalunya).
[4] This big keyword includes proposals over *Public key*, *Elliptic Curves*, *Symmetric algorithms*, and *stream cyphers*. Other keywords mention are the *Homomorphic encryption* and *secret sharing* but they are not the centre of this paper.
[5] see tango-controls.org for further information about the consortium.
[6] Synchrotron is a circular particle accelerator design to generate electromagnetic radiation (specially in the region of the X-rays but not only). This light is used in scientific experiments
[7] Neutron sources are scientific facilities with nuclear reactors or particle accelerators designed to emit neutrons for experimental studies

operating systems, having the computing sources of a very big computer with small embedded systems in the same network. When security is introduced, it must not "disturb" or it's easily deactivated. Then the solutions shall work fine in the smaller case, and *Elliptic curves*, *lightweight symmetrics* and *stream cyphers* are the best candidates on this business.

TANGO[12] is a Free Software implementation, divided in 2 parts (as will be explained in section 2). The core is distributed under *LGPLv3+* license and the device servers distributed by the *Consortium* are licensed under *GPLv3+*. There are other satellite tools in TANGO that helps to cover more area and extend the usage. This tools are also distributed under *LGPLv3+* licenses and they are ATK, SARDANA[8], TAURUS (to help on the SCADA behaviour) and MAMBO (that covers the feature of the data archiving). The auxiliary tools used inside the TANGO implementation are also Free Software as it is the OMNIORB[1] in *LGPLv2+* and ∅MQ[2] is *LGPLv3+*.

It is very important goal to have the TANGO implementation as Free Software, as this paper cryptography outcomes must be to have public access with auditable algorithms and sources.

After the introduction this paper is divided in 3 main parts. The first is section 2 to introduce concepts and definitions about *industrial control*, *distributed systems*, *information security* and *security engineering*. As this paper deal with the case of TANGO section 3 puts in context those previous definitions in over this particular case. Once all those things are on the table, section 4 can describe the first solutions that can be introduced on this system, together with others, more advance, that are candidates for next stages of a *cryptographic enginering* solution for TANGO.

2 Concepts

2.1 Industrial control

Largely used in the industry, a *Distributed Control system* is a particular case of an *Industrial Control System* and is a particular case of *distributed computer network*. Lets start from the industrial side for the definitions.

Definition 1 (Wikipedia (en)). Industrial Control System *(abbreviation ICS) is a general term that encompasses several types of control systems used in industrial production, including* Supervisory Control And Data Acquisition *(abbreviation SCADA) systems,* Distributed Control Systems *(abbreviation DCS), and other smaller control system configurations such as* Programmable Logic Controllers *(abbreviation PLC) often found in the industrial sectors and critical infrastructures.*

In this definition 1 are nested 3 concepts that must de also defined, and lets start from the smaller and then continues for the next little bigger.

Definition 2 (Wikipedia (en)). *A* Programmable Logic Controller *(PLC) is a digital computer used for automation of electromechanical processes, such as control of machinery on factory assembly lines, amusement rides, or light fixtures.*

With PLCs a production line can be controlled and monitored. This systems, even they are thought for local uses, often have telephony *sim* cards and are exposed throw insecure environments. Above those PLCs, the common is to have *Graphical User Interfaces* (abbreviation GUI), and them are also the next level in the ICS.

Definition 3 (Wikipedia (es)). *A* Supervisory Control And Data Acquisition *(SCADA) is a computer software to control and supervise industrial process remotely.*

By this definition of SCADA, many more computing elements can be introduced. As mention before, below SCADAs can be one or many PLCs, but also other computers with other very specific tasks, like specific measurement cards plugged or FPGAs[9]. Once here a higher level abstraction can be found when goes to a larger scale:

Definition 4 (Wikipedia (en)). *A* Distributed Control System *(DCS) is the computer software for a manufacturing system, process or any kind of dynamic system, in which the controller elements are not central in location (like the brain) but are distributed throughout the system with each component sub-system controlled by one or more controllers.*

[8] See sardana-controls.org for further information about this collaboration.

[9] FPGA is the abbreviation Field-Programmable Gate Array even is widely used in Controls, here is not have its definition because it is not considered an element of an ICS, but can be comparable to a PLC

2.2 Distributed systems

From the *Distributed Control System* definition (4) has been used introduced the concept of a non-centralized system over a network. The concept of the distributed comes from the time when computer networks becomes more and more common and the idea of one processing unit with peripherals become expired. Continuing with the view of the industrial control systems, a PLC is what can be sawn more similar to a single *personal computer*, but with the networks born the idea of many pieces doing related and/or independent tasks but coordinated. From this concepts emerges the definition of a *distributed system*:

Definition 5 (Tanenbaum [3]). *A* distributed system *is a collection of independent computers that appears to its users as a single coherent system.*

Every single element, called an *agent* of a distributed system is independent of all the others, but its meaning comes from the interaction with those others (and obviously some of them with the user). This independence is what gives one of the most special features to a distributed system: the scalability. Because there is no central point, neither the agents depends hardly one each other, the growing number of agents depends only on the infrastructure capacity. Perhaps more cpu can be required, perhaps more bandwidth, but the number of agents has virtually no limit.

Once a system likes to receive the *tag* of distributed some conditions must be complained. Continuing with the Tanenbaum's explanations, there are several *transparencies* that a candidate for distributed should complain. They are refereed to characteristics that shall be hide to the user of the system. But every rule can have exceptions, then one can define degrees of compliance with these transparencies. Not all the distributed systems respect all these transparencies, neither in the same extent.

Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide a faulure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

Table 1. Transparencies in distributed systems from [3].

In table 1 are listed 8 transparencies that a system should complain to be called distributed, but as mention not all the distributed system follows all of them neither in the same degree. Depending on the requirements of a distributed system those transparencies will need more or less degree of presence. It is not the same a mobile agent system than a DCS. Specifically, in the case of DCS in the industry, a highly specialized hardware would be only controlled from an specific computer that has the specific card plugged (often expensive). This happens most of the time with Tango, but it does not mean that other agents can be moved from one machine to another, when they do not have requirements like the mention before (for example a cpu intensive processes).

Although the first idea of transparency seems broken by hardware restrictions (one may think in the *location* transparency), this can be wrong. Imagine what would happen if this specialized hardware fails. If the agent can be moved to new location (autonomously) and from there report this failure by itself, the consistency of the system will be greater that to realize of this failure because the agent is not available.

Detailed description on Tango's transparencies will be set up in section 3.

What allows all this agents in a distributed system to be relocated, replicated, and so on with the transparencies is what gives coherence to the system, and this receives the name of *middleware*.

Definition 6. *The* middleware *is what supports heterogeneous computers and networks while offering a single system view.*

The middleware is nothing by itself, it is nothing *tangible* in the system in the meaning that the user can not interact with. The middleware is the environment where the agents live, is what stablish the rules that all the agents will follow.

2.3 Information security

At this point it becomes clear what is a Distributed Control System. Now, it is time to define other key concepts for the purpose of this paper like it is the *information security*.

Definition 7. Information security *is the protection provided to the information in order to minimize the risk that it becomes compromised.*

By this definition 7 many risks has to be consider like unauthorized access, any modification and also the destruction of the information to be protected. Following this risks the principles of the information security are listed in the table 2. There are the 5 basics and well know, and has been added some extra that will be needed for the purpose of this paper.

Confidentiality	Information must be disclosed only to the authorized.
Integrity	Only authorized can set in the system and no one else can modify.
Availability	Information must be accessible for those who are authorized.
Authenticity	Information must only be emitted by the authorized.
Non-repudiation	Forbid validity changes on the information emitters.
Auditory	Trace who access where (extremely useful for a security breach analysis).
Third-party	Protection to averting damage.

Table 2. Basic principles of the information security plus extra non-basic objectives.

These last elements about auditory and damage control, are like many other things in security, they are double-edge swords, specially in the auditory case. Always depends on how to install an specific system, to decide if those principles clashes with privacy. Perhaps that is why they are not *basic*s.

Privacy is not against security neither the other way around, but the difference in this auditory principle is who does it and under which circumstances. When is the same the owner who audits itself, with transparency with the users, in order to find if someone is acting improperly, this can be a good tool. But this tool can be also used to spy the behaving of the users.

Further than secure the information itself there is an environment, where the code runs, to be secured. In a distributed system besides the fact of provide the basic principles of the information security, one have to step in the trustiness of the machine where this code is running. From the example of the multi-user machines, where one had to be aware of the other users, in a distributed system there are other agents (perhaps with the same user identification and permissions) running in parallel. The security in the agents communication is not only a matter of networking (the channel), many other layers are involved.

2.4 Security engineering

Develop security engineering is a multi-disciplinary field. It is not only a matter of knowing the mathematics behind the cryptology, it also requires a knowledge on networking and hardware behaviour, and very often the psychology to understand how the things applies in practice. It is use less to implement de hardest cryptosystem if this implementation is revealing information in a side channel, or to have the strongest password policy that forces the users to write them on papers because is not humanly possible to remember.

TODO: "*Security certifications:* ISO 27001, SSAE 16, PCI DSS 2.0, HIPAA compliance."

This field makes feel like a bad guy. Everyday training is required and the best way is to think over all the aspects of the live, from the insignificant to transcendent. How to protect a pet shop if you do not allow yourself to think how to steal on it? You do not have to steal it but is not impure to think how one that likes to do it can behave as.

In the book "*Security engineering [4]*" three[10] scenarios are highlighted: A *bank*, a *military base* and a *hospital*. Why this three and not others? Probably because with them all the critical assurance requirements are covered. The fundamental on each scenario is the point of view, and any new scenario can be described in terms of different degrees of the characteristics on those highlighted. Any new scenario can use this to define which is the current protection problem on it.

[10] There is a fourth scenario in the book, but is more a corollary of the previous.

Inside a bank: In this case what must be clear is the goal of the security: protect information and avoid deceptive transactions. Then the information and the capacity to do these things must be uniquely in the hands of the stuff. But even if there is trust with staff, is a good procedure to dilute the authorizations when the operation has some degree of unusual or dangerousness. To any operation, a degree (or degrees) of different risks can be tabulated. With this information, a single operation can require some level on the person who is doing or even require the approval from more than one person. This also includes current web applications systems where the staff are not directly involved, but the client gets some staff roles.

Another situation described in this kind of scenarios is the monitoring of activity patterns. There are also degrees on privacy invasion, or decay in a need to monitor the one who is monitoring. It is not necessary to go to much in the detail, on the opposite, with enough abstraction, anomalies can be raised and detect, for example, when in one office there is a robbery.

Looking on a military base: This scenario is an example on where complex systems interact with countermeasures (and often *countercountermeasures* and so on). The desired image from this kind of places is like if it has a dome that looks transparent when it really like to not show anything at all. Not going to the field of the *steganography*[11], the purpose of security is to distinguish between *friend or enemy* (FoE). The keystone on this scenario is the trust on the incoming information (assuming your own computer is not compromised).

Another characteristic of this kind of scenario to highlight, is a extreme trust in the chain of command, but together with an also extreme requirement of solutions in case of a *head-cut* situation.

A day in a hospital: A completely different scenario than the previous is what happens with patients records. The access to the information has to be granted to some specific staff in the precise moment that they need to, but this can expire in a while. When some one is receiving treatment, the staff should have access to all the available information, but not when the treatment is done. Later on time, the information should be still available but under an anonymised format to allow medical research without compromise the privacy.

Should not lose sight that an scenario like a hospital is a place where new people is coming in and out quickly, where the staff has to drop what they are doing to run into an emergency when the less priority is a simple act of locking a computer.

In all this case the main issue is the misunderstanding of security from the heads and administrators. When security measures are understood as arising from cowardice. When security is rejected by considering it as simple paranoia.

In security engineering there is an important concept, introduced in [5] under the name of *practical paranoia*. There are security decisions during a distributed system design and implementation, but there are also other decisions related with the time when this system is placed somewhere, put in practice. Then is the moment to respond two extremely basic questions:

1. How much value has the information?
2. How much resources has the attacker?

These questions have reference to define how high should be the wall you build using cryptology. There must be a balance between how important is the information and the cost of protect this information. Often that is not easy to answer, from one side because the value of the information can be objectivated but with a subjective component, and the resources from the attacker are estimated (any of the potential attackers should not show the real potential), and the attacker's motivation must be also take in the equation.

One may think, these two questions can be reduced to only one by assuming an attacker with infinite resources. Assuming an attacker with a computational capacity estimated on the best machine in the world and increase its magnitude order, is not practical because this attacker would generate this secret information by itself with a portion of the energy estimated in the attack. Keep in mind and never forget that the weakness point in a security system are the people involved. Perhaps is good to have on mind, that when someone likes to steal a car, the target will be the easiest, but you are in trouble when this *villain* really wants specifically your car.

[11] Steganography from wikipedia: "the art and science of writing hidden messages in such a way that no one (...) suspects the existence of the message"

The last element pointed by paranoia, is the *ego*, previously referred as the subjective component. The consequences of a secret revelation must be cool mind evaluated. Yes, public image is important, but is also a bad image when something is protected with a digital locker that grants the protection of the contents for the next decade, when the information will become public in a month.

In the book "*Cryptography engineering*" [6] another interesting concept is introduced: the *thread model*. And the section about it starts with great words:

Definition 8 (Every system can be attacked [6]). *"There is no such thing as perfect security. The whole point of a security system is to provide access to some people and not to others. In the end, you will always have to trust some people in some way, and these people may still be able to attack your system."*

As from the mathematical cryptography there is the definition of *perfect secrecy* as the goal towards which have a tendency in an asymptotic manner, almost the same thing happens with *perfect security*. In practice, a *thread model* is a balanced equation between interests and threads.

Set up cryptography in a system to secure it is the easiest part even it is not an easy task, as we will see in section 4. The hardest part is when the system is used in practice, where human intervention allows side channel attacks, or misunderstandings on the configuration that shutdowns the cryptoghraphy tools with out notice, or many more similar issues.

2.4.1 Security levels Even the fact that *every system can be attacked*, this do not leads the a dead-end. Two systems requires different levels of effort to be breached in an *exhaustive search attack* then different levels of secrecy can receive different level of cryptography. Recently Lenstra, Kleinjung and Thomé [7] have described in a very graphical way the security levelling, based on the the equivalent requirement of energy between break a system and boil water.

The introduction in TANGO of the security levels, when introduces security, can start with the European reference on this matter, the "*fiche 17*"[8] of the *European Commission* (Other world regions can have different references like USA have FIPS 140-2 [9] that should be also a reference in developing an new institution levelling). This document unifies the description of security levels using the *Europol* as reference:

- *Open or unclassified*: not necessarily public access, simply information that will not cause harm is anyone knowns.
- *Restricted*: unauthorised disclosure could be disadvantageous.
- *Confidential*: unauthorised disclosure could harm essential interests.
- *Secret*: unauthorised disclosure could seriously harm essential interests.
- *Top Secret*: unauthorised disclosure could cause exceptional grave harm to the essential interests.

Any institution that stablish security levels for its needs, can decide to have a different number than 5 levels and for different reasons. For example they can be only 3: *short-term*, *medium-term* and *long-term* as *life span* is mention in the model proposed by Lenstra and Verheul [10] in 2001, together with the degree of infeasibility of an attack, the computational resources and its evolution in time and the cryptanalysis improvements. They can be 2 or they can be 10, but what for sure should be is well selected the algorithms to be used. One has to be very careful selecting the appropriate key sizes, does not have sense to use an incredibly big elliptic curve together with a subpar hash and a symmetric encryption weakened due to performance restrictions. As also Lenstra have explained in [11] the security level is defined by the weakest of the three algorithms.

There is another element to consider that the solutions proposed in section 4 will allow. There is the concept of *compartmentalize* and even if two sets of information lives in the same security level, they may need to be isolated as much as possible to the attacker point of view. It will be mention specifically in the public key setup, but already can be said that using different elliptic curves over the same finite field, an attack over one of them is use less against the other, giving a new powerful tool of segmentation in between the same security level.

3 Tango characteristics as a DCS

As TANGO is a Distributed Control System (definition 4), even if it has skills of an SCADA (definition 3) for particular user cases, the elements defined in section 2.2 shall be glimpsed. What

is known as the TANGO-core is the part of the system that follows the definition 6 of the middleware and it is what provides the necessary homogeneity to the agents in the system shown coherence.

In this distributed system, the agents are called *Device*s, each device has one and only one class (called *DeviceClass*), and runs inside a process in an specific machine (called *DeviceServer*). Figure 1 can clarify the schema but in words can be said that one *DeviceServer* can host many different *Device*s, being them from the same *DeviceClass* or not. This provides the flexibility to the system for inter-device communication can be setup using the middlelayer of internal process communication like shared memory or message passing.

The *Device*s are the agents of this distributed system and they follow an *object oriented* (OO) paradigm. A *Device* is an agent if you look TANGO as a DCS, but from the abstraction to see them as an application, the *Device* can be sawn as an object. As an object it has some encapsulation grant.

From this encapsulation, lets start with what is called *Properties*. They can be sawn as the parameters in the constructor of this object, and also they are the tool to store permanent data with in the system. Any object has *attributes* and *methods* in the object oriented design and in TANGO they also exist.

The *attributes*, in *OO* are often private and non accessible as interface to the object, in TANGO they are public and are the key to see how the things are going. These TANGO *Attributes* can be read-only or read-write (and other modes rarely used like write-only) and all of them have and specific type and dimension. Those characteristics are immutable, and once it is build it cannot change in execution time. As can be read in figure 1, in the box of "*data types*", the list of possibilities is not too long, and as will be explained later they are independent to the architecture where you run. Attributes can be from the simplest boolean to many sizes of integers (8,16,32 and 64 bits), the case of floating point is mapped the basic CORBA types with a *DevFloat* of 32 bits and a *DevDouble* with 64. And finally there are two more data types: the string (or list of characters) and the encoded. Explain the *DevEncoded* is out of the scope of this document, but simply mention that has a header that explains how are the data stored on its body.

The methods of this object are the *Commands* that represents actions to be take. This methods can be parametrised with an input argument, and the responses as output argument. There is only one *argin* and one *argout*, but it can be an array instead of a single element.

The figure 1 shows a box with the data dimensions, but this is applicable for the *Attributes* of the *Devices*. Any attribute has a unique and fix type and dimension. An *scalar* attribute contains only one element, when the *spectrum* has a finite array ($1D$) and the *image* is a limited length array of limited length arrays ($2D$). The data types for *Command* arguments and *Properties* are a bit different; *image* dimension is not available, and *spectrum* types are *scalar* types as *DevVar*Array* (except the *DevEncoded*), plus *DevVar[Long,Double]StringArray* that is a list of pairs where each number is linked to an specific string.

3.1 Tango naming

In a distributed system, one of the most important things is the *naming*. In the communication with the agents, and the communication between the agents, they have to know the name of each other to have the possibility to *talk* between them.

All the agents in TANGO have a unique name structured as a string similar to the web services. The more generic name for a *url* is a *Uniform Resource Identifier* (or URI). A *Uniform Resource Locator* is what enables in a system to locate a resource. In the common use of the web, a *url* identifies an specific resource on the referenced website. A TANGO *uri* can identify a *Device* with:

$$[\texttt{tango} : //\texttt{facility} : \texttt{10000}/]\texttt{domain/class/member}[\text{–}\texttt{NN}[\text{–}\texttt{sufix}]] \quad (1)$$

The protocol can be assumed and the same with the database server denoted as *facility* with the port access to the database, that can be set up bu environment variables. The reduced way to name a *device* is by `domain/class/member` and is giving the possibility to stablish a hierarchy of three levels in the naming.

The basic access brick is the *device* and as it has been mention, it is the agent in the distributed system. Based on design patterns on of the objects you can request to middleware is a *DeviceProxy* and to build this proxy you need the *device name*. But there is a secondary element of this device that the middleware provide a proxy, it is the attribute. Each of the attributes can be accessed

as an encapsulated object called an *AttributeProxy*, and the naming for it is to append another element in the naming tree:

$$\texttt{domain/class/member[-NN[-sufix]]/attr} \tag{2}$$

The other important element in the normal use of a device are the commands, but they are not encapsulated apart of the proxy, and they do not need a naming in the URI tree. It was not mention before to avoid confusion but both, device and its attributes have properties. They are permanently stored using the database, and is the middleware the one in charge to access to read or change each of them. They neither are included directly in the URI naming tree but in this case, the TANGO manual [12] have setup a way to call them:

$$\texttt{domain/class/member[-NN[-sufix]][/attr]->Property} \tag{3}$$

In this case, the protocol and database server name has been avoided for simplicity, and notice that the attribute part is optional to remark that the naming for a property has the same structure even if it is refereed to a device or to an attribute.

TAURUS , that uses much more the naming to setup models in Qt, has extended the naming to support other elements that needs references. What this is about is the attribute configurations that are not attribute properties but characteristics like can be the unit, the label, the format, and so on.

$$\texttt{domain/class/member[-NN[-sufix]]/attr?configure=unit} \tag{4}$$

Once a name is known, the way to contact the destination is based on the TANGO's '*phone-book*', what is located in a MYSQL database in a machine that has been labelled in the previous example 1 as `facility`. Even if it is a human user from the presentation layer or another agent from inside the domain layer, all has to request to the middleware access to the TANGO-db to locate where is running what is named and to which port will have to address the contact. This TANGO-db is a centralized element on the system.

In this database there is another valuable information stored. The database is the permanent storage of this distributed system. As has been mention before the *Properties* are the unique element stored outside the *devices* and they are used to configure these *devices* on the *DeviceServer* start up.

Last of the details to mention from TANGO is the logging system. It is very simple and consists of a series of files stored locally by the agents in the machine where each agent runs. As the machines where the system runs are inside a trust enviroment the access to this log files is guaranted but anyone with access to the machine can read those files. The content on this files depends on the agent developer and the use of the streams with different logging levels available and when a *device* process (the DeviceServer) is launched, in the command line the log level can be set up.

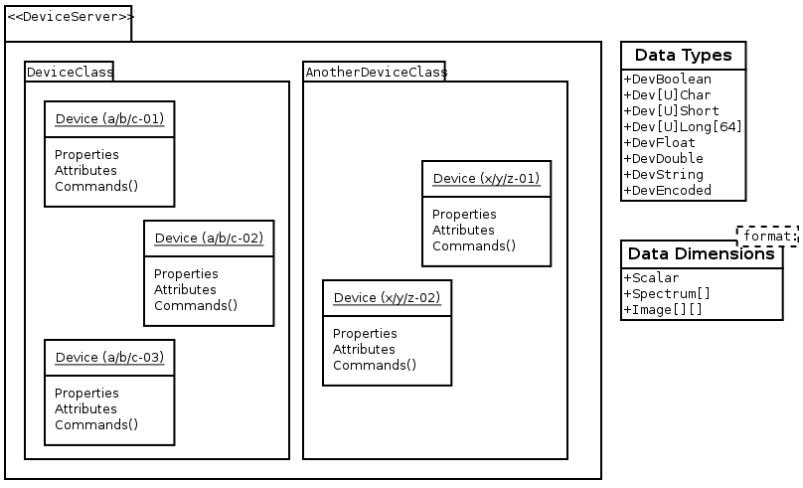

Fig. 1. TANGO agents process structure FIXME: "*To be improved*"

3.2 Tango transparencies

Now, with this knowledge is time to move back to the table 1 about distributed system transparencies. As TANGO is a software built above CORBA, it is using it to deal with the degree of support of those transparencies.

The 8 transparencies shall be divided in 3 groups: the ones provided by the middleware tools, the ones that depends on the code in the agent, and the ones not present in TANGO.

Transparencies provided by middleware: The first group, starts with the *access* transparency. It is because of the CORBA common data types in the *IDL* (Interface Definition Language) files that the data types are homogenized in a tango system. Forget if compiler set an `int` of 16 or 32 bits, a DevShort will have 16 every where and DevLong, 32. Next, *Location* transparency is not supported by the middleware, every search for an agent (for example, by name) will require an access to a TANGO-db, that one can see it as a '*phone-book*' of agents and this is a *centralized* weak point. Any issue on the database server makes you blind in the system. Even is the agents continue working you would be unable to see it, and the performance of the system decays when agents are trying to communicate with other agents.

Transparencies provided by agent developer: The second group of transparencies are the ones that depends on developer sensitivities. In this case, there is no supervision about if it supports it or not. The first transparency that depends on the developer of an specific *DeviceServer* is the *concurrency*. The way the specific devices does the job when an operation is requested to them, is only managed by the agent developer. From the code in *SourceForge*[12], there are more examples of agents that does not support concurrency than examples to follow in a new development. This is very related with the *persistence* transparency, because again, the code to access an specialised hardware is embedded in the agent code. To often a user can not distinguish a hang agent than a hanged instrument. And again this links to the *failure* transparency. In my humble opinion, the issue comes from a non-well undestood of *extreme programming*[13] philosophy when there is no real time in code review and unit testing. Then too often, when a *DeviceServer* is launched with the hardware not present, instead of a notification from the *Device*, the situation is a premature death of the agent.

Not supported transparencies: And the third group are transparencies not supported at all. *Migration*, *relocation* and *replication* are transparencies not supported by TANGO. But despite a weakness it is a design choice. The agents cannot move their code from one machine to another while you use them. When a device is restarted, event in the same machine, it has lost any request you made to it. When *migration* is not supported, *relocation* even less. *Replication* is a task to be made by humans and agents have not capabilities to decide anything about that.

3.3 Attacks

Previous references to books in section 2.4 let seen between practical attacks to a distributed system like TANGO. But event this type of system have some peculiarities, there are many things that can be reused. In a widespread distributed system, the agents communication is not too different than RFID card/reader communication [13]. The figure 2 shows an example of the agents involved in a normal operation of the Control System. For example, in the centre there is a representation of an agent in the distributed system in charge of the alarms. This is a key device in charge to do the monotonous task of check that everything is running as expected. This is a good example because this must be able to talk with every one that is being monitored.

First type of attacks to any system, are the passive *Eavesdropping*. An attacker can do almost the same than this agent in charge of the alarms but not to advice about problems, but something else like record private information, like can be data from a scientific experiment or industrial secrets.

There are other type of attacks that passes the line of the observation and starts to introduce actions to the system, and that is why they are called *active*. There are many different ways to do this tasks. The most known is the *Men-in-the-middle*, with good skills when the two participants

[12] See tango-ds.sf.net
[13] See Extreme programming for further information

to the communication, and because they do not know each other (no authentication), have the communication bridged by some one else in the middle that is repeating the communication from one to the other and all the information is accessible to this Eavesdropper. This attack is just half a way between active and passive, but often is combined with the other type of active attacks.

The active attacks can be splitted in three main categories: avoid the information transmission, avoid the reception, modify the transmitted information. For the first case, adding enough noise to the communication channel you can block the transmission of data from RFIDs and in this case you may not distinguish from when there are not cards. In a distributed control system the situation is different because, the environment are better known. The disappearance of a device will be perceived. But there are other degrees of this noise attack, perhaps with only poisoning the information the attack can reach the goal. Imagine a device that is raising an alarm, but the attacker is interrupting this. Even further the attacker can *spoof* (modify or send an old message with a new time stamp) the data and the receivers still think that everything is going well. Still more beyond, the alarm device can be completely supplanted or tampered and build information from inside the system like one more of the agents.

It is mandatory in any system that likes to practice security, not only to have the necessary tools to protect itself to the attacks, but the tools to detect if something has crossed the barriers. As was been mention about perfect security in 2.4, do never have an *ego* that makes you thing no one can cross the defences. But detection is nothing if there is not recovery. Never forget the social engineering against supervisors, and left the first line of this work to *auditor agents* that, in case of detection, can start the fight and the humans get notice to monitorise what is happening.

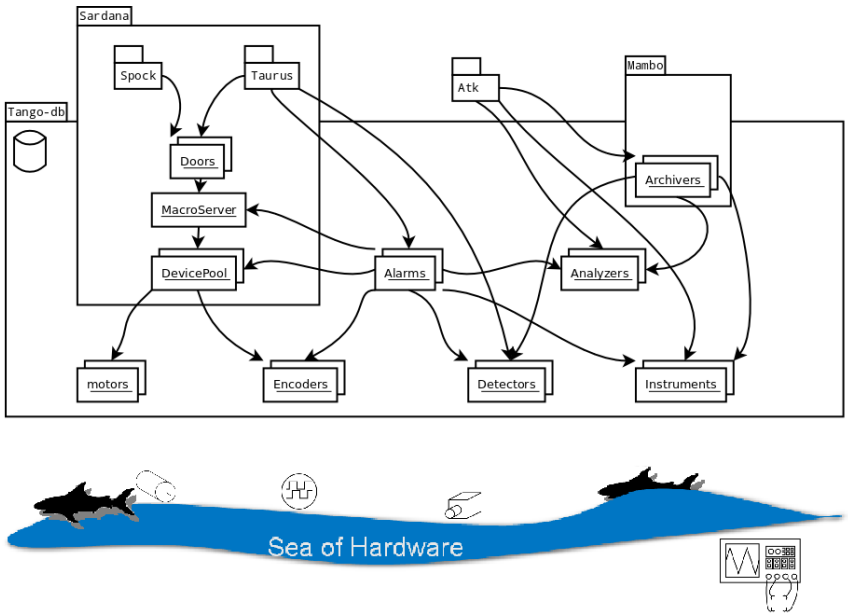

Fig. 2. TANGO schematic layout FIXME: "*To be improved significantly*"

4 Solutions

Although in security engineering section 2.4 the pointed scenarios can lead to think in solutions like *fingerprint readers* or *iris scanners*, the purpose of this paper focus on agents interrelations inside networked computers, and is not about human access to the system. But once a user is authenticated it can act like another agent.

What TANGO needs as final product that introduces security, is to add to the URI naming an '`s`' of secure in the protocol name like web had with "*http*`s`" or many other service like "*smtp*`s`", "*imap*`s`" or if need be, how the "`ssh`" has replaced the "*telnet*". There is not enough activating the `OpenSSL` flag in the OMNIORB compilation, there are levels of abstraction above CORBA that must be covered. TODO: "*What about the ∅MQsecurity?*".

Another fact has been mention in section 2.4 and it is about the issue that any agent can not be trusted if the machine where this agent runs can not be trusted. It does not have any sense to trust in an agent that is carrying a secret key that may have been compromised during an execution inside a machine that is out of our trust ring. But this can be necessary and perhaps can be helpful to have some '*trust*' flag like in the *OpenPGP Public Key Infrastructure* (abbreviation PKI), but from the *Certificate Authority* (abbreviation CA).

Once said this points, it is time to go inside of the proposed solutions for TANGO. Here we have two main solutions in the next subsections to complain with the basic principles of *information security* (see table 2 in section 2.3), followed by two introductions to advance solutions.

4.1 Key Managing Infrastructure

TODO: "..."

4.2 Use of *Elliptic curves* for authentication and public key encryption

The TANGO DCS runs its agents in very different kinds of hardware. It is possible that an agent runs in a huge server with more resources that it could even use, and the same agent later on is moved to an embedded board where the resources are extremely limited. Or perhaps the user interface runs in a good workstation with genuine resources, and there must not be differences between device running in big or small machines. Remember this is a requirement for distributed systems transparencies (focus in *access* transparency, see section 2.2 table 1 and specifically about TANGO in section 3, these are tools that the *middleware* must provide).

Elliptic curves are a very good tool to be helpful with this scenarios where embedded system coexist with others. But there is another feature of the elliptic curves that can be useful here. Remember in the *security levels* (section 2.4.1) where the idea of different systems that shares the same levels should be isolated between them, and this solutions is helping this requirement. Because with a different elliptic curve per subsystem, the cryptanalysis on each is different, even if they are defined over a finite field of the same size (even when they share the same prime), they can do this compartmentalized.

Nowadays there is a big discussion over this possibility of the curves use. The beginning of the standardisation there where two tendencies: allow any curve or restrict the number of curves. Focussing on the ANSI X9.62 recommendations [14], not only restrict the set of curves is bad, but too many keys sharing a curve are a thread by itself. For this purpose the use of the NIST curves is not enough (they have been published in 2000 with in the annex 6 for the FIPS PUB186-2 updated in its third version [15] in 2009) even the curves are taken from a bit bigger list, like [16]. The standard for *OpenPGP messages* [17] restricts even more this set to only 3 curves for all the world over 3 security levels on the standard. When in another hand, the standard for *TLS* with curves [18] allows any curve from certain formulations of the curves over $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$.

The final solution must not restrict to a set of curves. Even if a new set of curves is added, like can be the *Brainpool* [19], it will never be enough for a world wide use. A smaller restriction of a set of curves described in a certain way like is the Weierstrass Reduced From (abbreviation WRF) is potentially also an issue if new sets of good curves are discovered like the ones proposed in [20] (Montgomery and Edwards formulations). It is possible to start with WRF, but extension capability is mandatory.

In fact, what can help to follow the recommendations from the ANSI X9.62 is to have a public auditable algorithm to allow each one to generate its own curve. Perhaps, in case this is a computationally expensive cost, all the keys within a subsystem of one particular security level, should have its own curve. This is a feature that is not easy to implement for solutions like *OpenPGP*, but is an environment where the PKI is build from scratch this is a feature that can be included.

On this public auditable algorithm, there is work on going [21] where is proposed solutions based on *Complex multiplication* or in elliptic curve *isogenies*. This work has been based in previous development of [22], [23] [24], and [25]. From this two possible solutions, one has an extra feature, because from the idea of a public key cryptosystem using isogenies [23] would be possible to introduce a new operation that any other system can not do: the *cryptosystem reset*. This feature could allow the system to renew the elliptic curve in use, by a newer one with the same size but where if there is any attack ongoing on the first curve, the cost to *migrate* the attack from one curve to the other will be bigger than start the attack on the second from scratch.

As mention, the *PKI* and the *CA* has to be designed from scratch to be inserted in the TANGO middleware. And this is a great advantage because this can save many of the design issues where the current infrastructure have limitations. It is a great opportunity to have nothing that forces backward compatibility on this point, but is a great responsibility because each decision on this matter will affect the later evolution. As TANGO has, and will probably continue having, the TANGO-db, this PKI can use it. But if someday, TANGO does one step further in distribution, a hard link on that would be a stone in the shoe.

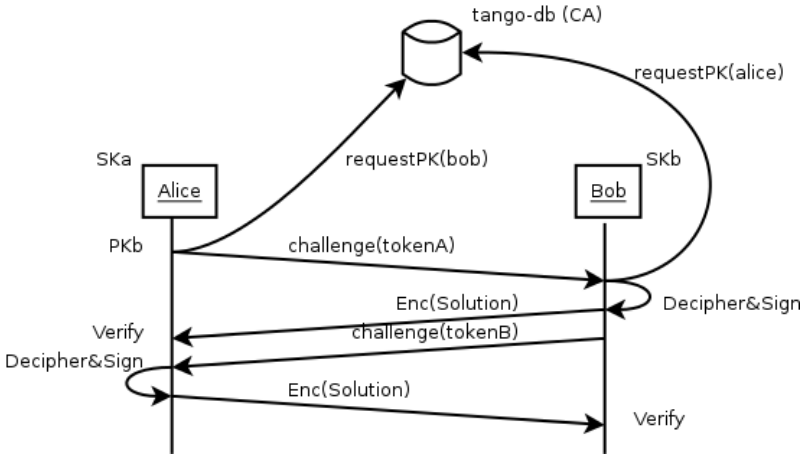

Fig. 3. Two agents authenticate each FIXME: "*To be improved*"

In figure 3 are a representation on how a double authentication between agents can be set up. It is not a unique solution but anyway it must be reviewed in the design and when implementation to avoid any of the known mistakes in authentication process. But this schema is good enough to show one initial problem to be resolved: *how to introduce a new agent in the system?* This is a keystone point because the first time the agent is introduced to the system, it has to be authenticated by someone. It is like the developer of the *DeviceServer* introduces to the system the *Device*s from the process *instance*. This paper does not go inside these protocols, but in the system start up for sure the *installer* must setup a set of developers (humans at the end) that are allowed to introduce agents to the system. These, as usual, introduces the same problem of the *certificates* and the solution can be similar to a *trust ring*.

4.3 Lightweight symmetric ciphers

Once a device is authenticated and there is a secure channel using asymmetric encryption, this can be used to exchange session key when an attribute likes to be accessed. This paper proposes two main ways: symmetric algorithms and/or stream ciphers. Both have *pros and cons* and a hybrid system would support both and use the best of them depending on the situation (with and agreement between the parts in the authentication). The key agreement should also agreed how the later communication will be (algorithm setup and key renew policy).

The constrains to select one way or the other must be the flux of data. The needs are very different when the transmission means a boolean subscribed that changes every few milliseconds, or if the transmission is a detector image of 6 mega-pixels, where each pixel is a *DevLong* (32 bit size), generated in the order of the hundreds of milliseconds. But the objective of this step is clear: made the transmission indistinguishable from *random*.

Block ciphers: The first of the solutions proposed for the information transmission is the use of lightweight symmetric algorithms. An important feature for this is the possibility to adapt the sizes of the blocks to be used, together with the encryption modes.

One of the most used algorithms for symmetric encryption is the *Rijndael*. This was the winner of the *Advanced Encryption Standard* (abbreviation AES) [26] that replaces the *DES* algorithm. There is a book [27] from the algorithm authors that explains very well the mathematics behind this algorithm. The original algorithm supports a block size of 128 bits and 5 key sizes (128, 160,

192, 224 and 256 bits) but when it was standardised, there where only used 3 of them (128, 192 and 256). But based on the mathematics behind it is possible to fine tune both sizes, the block and the key. There is a work in progress on this field [28] that is trying to adapt the *Rijndael* to be used as lightweight symmetric and also with sizes above the originals.

In the context of this paper, it is important to have smaller block sizes because the attribute types (section 3) are smaller than the basic block size and using directly the AES will at least duplicate the information to transfer for each value.

About the other size change, the key, can be useful the feature to have much more key sizes in order to have it in correspondence with the related security levels mention in section 2.4.1. The paper mentioned there [7] about the key sizes remarks also that it is mandatory to have correspondence between the three elements in a pack of security: public key, symmetric and hash[14] sizes. This sizes must be selected with utmost case, because the weakest is our incoming invitation to the attacks.

To use a data block of exactly the same size of the information to be encrypted could be a wrong solution. For data integrity could be good to include a *padding* to the information with an extra information there. The data could have an structure that can reveal information even encrypted, for example if there is an integer of 64 bits long but it is a counter that the attacker knows that it has started recently. In this cases the *padding* is useful because it adds unpredictability to the data sent.

In the case of the arrays of data, like *spectrum* and *image* dimensions ($1D$ & $2D$), the data to be encrypted can be too big to fit in a block size even if we try to extend this size. The waveform of an oscilloscope or a picture taken from a camera, must use an upper layer in the encryption. The block size can be the size of the elements data format, and help with the *block cipher modes* to operate with the complete set of information to be sent.

There are several modes. Starting with the Electronic CodeBook (abbreviation ECB) that is insecure at all because it has not a good randomness and, for example the case of an image, the structure of the data maintains the structure and the view of the ciphertext suggest the original image. Others like can be the *Cipher-Block Chaining* (abbreviation CBC), *Propagating Cipher-Block Chaining* (abbreviation PCBC), *Cipher FeedBack* (abbreviation CFB) or *Output FeedBack* (abbreviation OFB) can be really good but they are not parallelizable. The option of the *Counter mode* (abbreviation CTR) includes the best options: does not requires an extra padding and is parallelizable. This mode turns the block cipher into a stream cipher.

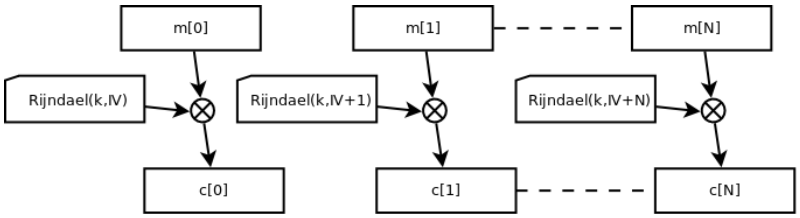

Fig. 4. Counter (CTR) block cipher mode.

The figure 4 shows a use of the CTR block cipher mode, but there is a prevention before use it. One must be absolutely sure that no *Rijndael(k,x)* is used more than once the the *IV* (known as *Initial Vector*) shall be choose with two parts: a *nonce* and the counter, being this *nonce* sufficiently large to be never repeated during the live time of the key.

Stream ciphers: There is another tool for the information transmission to be explored. This has the advantage of non block size limitation. One can see it as one particular case of symmetric systems. As it is a very often system used for broadcasting information, it is a candidate for the multicasting feature of the events emission using the ØMQ. There is a field in cryptology, called *secret splitting* that focus in this multicast feature.

Basically a stream cipher is a *Pseudo Random Generator* (abbreviation PRG) and is very tricky because the key size choice is not the only restriction on it, there is also the live time of the key.

[14] In this paper are not included the hash algorithms as like as the symmetric algorithms are due to an scope reason, but with them it happens a similar thing than with the *Rijndael*.

The source of this systems are *Key Derivation Funtion*s (abbreviation KDF) used as PRGs to `xor` the key flow with the data flow. As this is almost a *One Time Pad* (abbreviation OTP) the system can be considered a *perfect secrecy* as define by *Shannon* in [29], if the tools below complain some conditions. But do not forget that is almost OTP and the only way to maintain it secure is to renew the key before the KDF can start to produce repeated sequences.

The main stream ciphers shows a limitation in the key size. Starting with the `Rabbit` [30] that has been only designed to 128bits (with and initialisation vector IV of 64bits), or `Chacha` [**?**] that has 256bits for the key. A third way is interesting. An algorithm called `Sosemanuk` [**?**] is a stream cipher that has nested inside a block cipher called `Serpent` that was one of the finalists in the *AES contest*. This perhaps would open the door to, once there is an internal implementation to use the generalised *Rijndael*, it can be used as part of an stream cipher.

But as has been said at the beginning of this stream section, the basic skill of them are that they are PRGs that, with simply an `xor` with the data are good enough to be something like a OTP. Then another option can be the use of the KDF of the *Rijndael* that has this feature to allow many key sizes due to the generalisation. But where again another option must be evaluated. There is an RFC [**?**] with an algorithm called AESWrap that uses the AES in 6 peculiar turns in the way that remembers the 3DES did with the DES. In this algorithm the key is split in such a way that this algorithm can be used as a KDF.

4.4 Further advanced solutions

Apart from the two solutions to introduce security in the communication between agent in a Distributed Control System like Tango, other features can be developed inside this system to provide security under other aspects of its operation. First of all will be highlighted the solution to introduce cryptography in the Tango-db, but one should not forget that this is a centralized element in a distributed system that, more than think in secure it, could be better to think in secure distributed alternatives.

There is another advanced solution proposed, basically thinking in one of the scenarios mention in section 2.4 (the bank remember the bank scenario). A desired feature for a Distributed Control System would be to have integrated with in the middleware the possibility to have *command authorization*, like can be to have the *acknowledge* from a subset of users (or even agents) from a set of many more.

4.4.1 Cryptography for the database The Tango-db, as has been described in section 3, is used mainly for two functions: a '*phone-book*' to locate the agents in the system and for the permanent storage of the *properties* that are, at the end, the configurations for the agents when they start running. Now a days there is no authentication at all in this part of the system, while it is true that the MySql database access can be password protected, the middleware access to this database allows anyone within the system to, not only query, but also modify this database.

A desirable feature can be to provide this database with access authentication once the agents (and the users) have this system proposed in section 4.2. But what about the contents? The solutions about encryption provided in section 4.3 are not a choice for database contents. There is a field in cryptology called *Homomorphic encryption* (also known as *Ordered cryptography*) that explores this possibility.

It can be too advanced to introduce this field here because first must be discussed the viability of the database as the centralized point of Tango. Could be better to start with an authentication to access the database (for reading but specially for writing) and focus the efforts in a distributed database where later concepts like the *homomorphic encryption* can be applied.

4.4.2 Command authorizations As mention in the scenario of a bank in section 2.4, a desired feature can be the dilute the authorization with more than one controller (agent or human). This kind of solutions can come from *Secret Sharing* solutions. The secret sharing schemas are based on the primitive that there are n participants and from them are required k to access one specific secret ($k < n$). This is called *(k,n)-threshold* schemas, and the interesting point is that the k required should not be established in advance. Any combination of k from the n participants will grant access to the secret, and this secret can be used for anything, for example in this case authorise an operation.

This is a field also to have on mind during a first stage to introduce security to TANGO. The authentication and encryption must be developed facing the situation where a secret needed to execute a command comes from a negotiation or from an schema that includes *secret sharing*.

5 Conclusions

As has been mention in section 4, introduce security to a Control System like TANGO is not as simple as activate the flag to OMNIORB to compile it using `openssl` `TODO:` "*And the equivalent in* ∅MQ". With this the middleware will have a very interesting access to many primitives in this free software, but the schemas where these primitives some higher level protocols must be developed and implemented for bidirectional authentication of the communication actors (always thinking that they would be agents or users). In the case of the data transmission, the `openssl` would be not enough. Some algorithms must be gently modified from the standards, to adapt to the specific needs of this case. And must insist on the idea of this finely adjustments, because they can break the maths behind, and there must be a consciousness revision to avoid side channel attacks.

After the basics for introduce security on this system, not forget that other elements can be weak (and the weaker will mean the attack). In this text some issues with the TANGO-db has been pointed, but would be good to explore other distributed solutions like the `UPnP` protocols to discover elements in the system and the permanent storage.

This has been only a start, highlighting what has been already isolated as possible thread and pointed to some solutions. But the fly to this destination will be long. Also this solutions will require a mind change on the development because, something like *debug on the fly* will be harder when the system have countermeasures against tampering. Code testing procedures will be required to the order of satellite navigation systems or similar.

References

1. D. Grisby, July 2009.
2. P. Hintjens, February 2013.
3. A. S. Tanenbaum and M. van Steen, *Distributed systems, Principles and Paradigms*. Prentice Hall, 2002. International Edition.
4. R. J. Anderson, *Security engineering - a guide to building dependable distributed systems (2. ed.)*. Wiley, 2008.
5. N. Ferguson and B. Schneier, *Practical Cryptography*. New York, NY, USA: John Wiley & Sons, Inc., 2003.
6. N. Ferguson, B. Schneier, and T. Kohno, *Cryptography Engineering: Design, principles and practical applications*. Wiley, 2010.
7. E. T. Arjen K. Lenstra, Thorsten Kleinjung, "Universal security; from bits and mips to pools, lakes – and beyond," 2013. http://eprint.iacr.org/.
8. "Exchange of eu classified information," 2003.
9. "Fips pub 140-2, security requirements for cryptographic modules," 2002. U.S.Department of Commerce/National Institute of Standards and Technology.
10. A. K. Lenstra and E. R. Verheul, "Selecting cryptographic key sizes," *Journal of cryptology*, vol. 14, no. 4, pp. 255–293, 2001.
11. A. K. Lenstra, "Key lengths," *Handbook of Information Security*, vol. 2, pp. 617–635, 2004.
12. T. T. team, July 2013.
13. S. Martínez, *Protocolos de seguridad para sistemas de indentificación por radiofrecuencia*. PhD thesis, Universitat de Lleida, march 2011. Directed by: Concepció Roig and Magda Valls.
14. "Ansi x9.62, public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ecdsa)."
15. P. Gallagher, D. D. Foreword, and C. F. Director, "Fips pub 186-3 federal information processing standards publication digital signature standard (dss)," 2009.
16. "Sec 2. standards for efficient cryptography group: Recommended elliptic curve domain parameters."
17. A. Jivsov, "Elliptic Curve Cryptography (ECC) in OpenPGP." RFC 6637 (Proposed Standard), June 2012.
18. S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)." RFC 4492 (Informational), May 2006. Updated by RFC 5246.
19. "Ecc brainpool standard curves and curve generation," October 2005.

20. D. F. Aranha, P. S. L. M. Barreto, G. C. C. F. Pereira, and J. E. Ricardini, "A note on high-security general-purpose elliptic curves." Cryptology ePrint Archive, Report 2013/647, 2013. http://eprint.iacr.org/.
21. S. Blanch-Torné, R. Moreno, F. Sebé, and J. Valera, "Security risk associated with multiple users sharing the same elliptic curve." Draft.
22. R. Moreno, *Subgrupos de Sylow de las curva ellíupticas definidas sobre cuerpos finitos*. PhD thesis, Universitat Politècnica de Catalunya, 2005. Directed by: Anna Rio and Josep M. Miret.
23. A. Rostovtsev, E. Rostovtsev, and A. Stolbunov, "Public-key cryptosystem based on isogenies," 2006.
24. R. Tomàs, *Volcans d'isogenies de corbes el·lítiques: Aplicacions criptogràfiques en targetes intel·ligents*. PhD thesis, Universitat de Lleida, march 2011. Directed by: Josep M. Miret and Daniel Sadornil.
25. J. Valera, "Volcales de ℓ-isogenias de curvas elípticas," *Sistemas Informáticos. Escola Politècnica Superior. Universidad de Lleida*, Sept 2011. Directed by: Josep M. Miret.
26. "Specification for the advanced encryption standard (aes)." Federal Information Processin Standards Publication 197, 2001.
27. J. Daemen and V. Rijmen, *The Design of Rijndael*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002.
28. S. Blanch-Torné, R. Moreno, F. Sebé, and M. Valls, "Generalised rijndael." Draft.
29. C. Shannon, "Communication theory of secrecy systems," *Bell System Technical Journal, Vol 28, pp. 656â715*, 1949.
30. M. Boesgaard, M. Vesterager, and E. Zenner, "A Description of the Rabbit Stream Cipher Algorithm." RFC 4503 (Informational), May 2006.
31. C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, (New York, NY, USA), pp. 169–178, ACM, 2009.
32. N. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes," in *Public Key Cryptography â PKC 2010* (P. Nguyen and D. Pointcheval, eds.), vol. 6056 of *Lecture Notes in Computer Science*, pp. 420–443, Springer Berlin Heidelberg, 2010.
33. M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?," in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, CCSW '11, (New York, NY, USA), pp. 113–124, ACM, 2011.
34. D. Stehlé and R. Steinfeld, "Faster fully homomorphic encryption," in *Advances in Cryptology-ASIACRYPT 2010*, pp. 377–394, Springer, 2010.
35. D. B. nad Craig Bentry, S. Halevi, F. Wang, and D. J. Wu, "Private database queries using somewhat homomorphic encryption," *International Association for Cryptologic Research*, June 2013.
36. J. Sen, "Homomorphic encryption: Theory & applications," *CoRR*, vol. abs/1305.5886, 2013.