COSMOS

Cultivate resilient smart Objects for Sustainable city applicatiOnS

Grant Agreement Nº 609043

D3.1.1 End-to-End Security and Privacy (Design and Open Specification)

WP3 End-to-End Security and Privacy

Version: 1.0
Due Date: 30/04/014
Delivery Date: 02/05/2014
Nature: Report
Dissemination Level: PU

Lead partner: Siemens
Authors: Leonard Pitu, Paula Ta-Shma, Vassilis Psaltopoulos
Internal reviewers: Jozef Krempasky (ATOS), Spyridon Gogouvitis (NTUA),

www.iot-cosmos.eu

The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement n° 609043

Version Control:

Version	Date	Author	Author's Organization	Changes
0.1		Leonard Pitu	Siemens	Internal draft
0.2		Paula Ta-Shma	IBM	Cloud Security
0.3		Vassilis	NTUA	Privacy
0.4		Leonard Pitu	Siemens	added first chapters
0.5		Leonard Pitu	Siemens	extended first chapter
0.6		Jozef Krempasky	Atos	Internal review
0.7		Spyridon Gogouvitis	NTUA	Internal review
0.8		Leonard Pitu	Siemens	Pre-final version

Annexes:

Nº	File Name	Title

Table of Contents

List of Figures

List of Tables

Table of Acronyms

Acronym	Meaning
API	Application Programming Interface
ARM	Architectural Reference Model
ADC	Analog-to-Digital Converter
D	Deliverable
DSP	Digital Signal Processor
FPGA	Field-Programmable Gate Array
GPIO	General Purpose Input-Output
HMAC	Key-Hash Message Authentication Code
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HW	Hardware
I2C	Inter-Integrated Circuit
ID	Identifier
ISO	International Standard Organization
IoT	Internet of Things
IoT-A	Internet of Things - Architecture
IT	Information Technology
IC	Integrated Circuit
JSON	Java-Script Object Notation
LDAP	Lightweight Directory Access Protocol
M2M	Machine-to-Machine
NIST	National Institute for Standards and Technology
PKI	Public Key Infrastructure
QoS	Quality of Service
REST	Representational State Transfer

SHA	Secure Hash Algorithm
SPI	Serial Peripheral Interface
SoC	System on Chip
SSH	Secure Shell
VE	Virtual Entity
WP	Work-package

1. Executive Summary

This report (D3.1.1) presents the preliminary version of the document describing the overall security architecture of COSMOS. COSMOS' main objective is the delivery of dedicated mechanisms which enable smarter "things" in the IoT domain. For this reason the COSMOS project is split into dedicated work-packages which converge to a unified smart platform. The activities described in this report were carried out in the framework of WP3 – End-to-End Security and Privacy – which provides a thorough security analysis of the cross-domain influences which COSMOS is affected by. WP3 uses this analysis in order to provide a dedicated security enhanced system architecture which serves the entire ecosystem and can be used as a root of trust by all COSMOS partners. This report is the first in a series which will describe, in more detail, as the project advances, the security enabled functionality of COSMOS, covering all functional and non-functional components, at module and at system level.

Together with WP2 – Requirements and Architecture – WP3 provides guidance for all other WP's as to how developers should implement their components in order to meet both performance and security criteria. Based on use cases and requirements set out in WP2, WP3 enhances COSMOS with dedicated security components which are designed to enhance the overall system and not to hinder performance of usability. Therefore, from the beginning of the project, WP3 decided to follow some of the established design guidelines which are common in security, namely the "IoT-A Architectural Reference Model" for the IoT platform level as well as NISTs "Guide for Conducting Risk Assessment" and BSIs "IT-Grundschutz International" for the overall security level.

WP3 started out to enhance WP2's output thus starting from the user requirements and key parameters introduced by this work-package. Technical considerations were of crucial importance as the needed performance and reference architecture of the platform were defined by these. As each component of the system architecture corresponds to a prioritized user requirement, WP3 carried out a detailed risk analysis associated with each component. WP3's activities were performed in parallel to WP2 in order to enhance the overall system architecture, with security enabled components and to meet the goal of having a "secure by design" platform. Using this approach also helped in re-analyzing the processes and scenarios described in WP2 but from a security point of view. In developing WP3, D2.2.1, which describes technologies and future trends and innovations, played a crucial role as the COSMOS platform aims for state-of-the-art and above technological level.

The present report serves as the first milestone in developing the COSMOS security enhanced architecture. As there are many open items, WP3 will try to solve the remaining questions over the course of the entire project. Thus, the present report presents the highest level of detail, possible at the present moment, concerning the security components of COSMOS. The interactions between the identified partners and sub-components are explained to the best detail possible. The present report reflects inputs collected from all project partners and synthesized into the generic COSMOS Security Architecture.

Further, WP3 will work on a more detailed system analysis in order to identify system interactions which need security enabled functionality. This analysis will cover all COSMOS' components and interfaces, as described in their dedicated specifications. Therefore the COSMOS security architecture will not only cover software but also hardware components which are used throughout the COSMOS environment.

2. Introduction

This section describes the main innovations and concepts which are used as fundamental building blocks for implementing the vision of COSMOS in the context of IT security. These concepts range from hardware enforced security to cross-system security based rules and guidelines as well as strong cryptographic mechanisms. The final goal is to use security mechanisms to enable the vision of COSMOS in a secure and trustworthy way which is easy to maintain and extend while offering the needed trust anchors.

Drivers for these activities are on the one hand side the protection against common security attacks and on the other hand side the need for privacy, as recently presented by the recent leaks covering PRISM and other “Big Brother” like programs. Present day security attacks not only pose high security risks but also provide insight into what the future holds: a world more security aware in which every flaw can be expected to be uncovered in a given amount of time. Attackers are motivated by material wins or hatred but still have limited resources as to state-like sponsored programs are much more difficult to fend off.

Developing sustainable smart cities applications requires mechanisms to ensure security and trust and preserve privacy. Such approaches should address both lower levels of IoT environments (i.e. hardware-coded techniques) as well as the data management and application levels as well as user management. Tamper-resistant smart devices, dynamic and evolutionary trust models, secure data stores, applications with build-in security and privacy are critical for sustainable smart city applications.

Personal privacy, trustworthy and security are of crucial important for both end-users as well as for system provider. Trust is lost fast in our ever evolving world especially in IT services. Privacy and security at an individual level are pillars of development activities and are a natural consequence of the proposed concepts.

COSMOS will facilitate IoT-based systems with end-to-end security and privacy, from hardware-coded approaches on the devices level, access control, encryption, multi-tenancy and cross-application mechanisms on the data level, to the IoT services level with the injection of privacy –preserving mechanisms within things themselves.

3. Overview

WP3 tackles security issues and as such, security and the respective attacks have to be defined.

A basic terminology for security and dependability was developed during the MAFTIA project where security breaches lead to the to the fault model described in [1].

Based on this fault model the causal triple fault-error-failure are:

- **fault**: adjudged or hypothesized cause of an error.
- **error**: part of the system state which may cause a subsequent failure.
- **failure**: occurs when the error reaches the service interface and the delivered service deviates from executing its function.

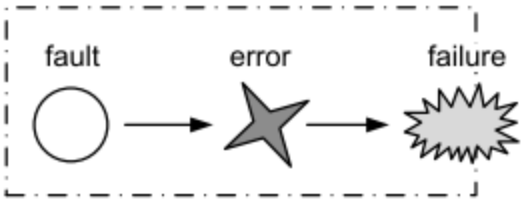

Figure 1 Fault Model

This chain reaction caused by a security attack, as depicted in Figure 1, describes how an attack propagates through a “set of subsystems that share one or more common resources.”.

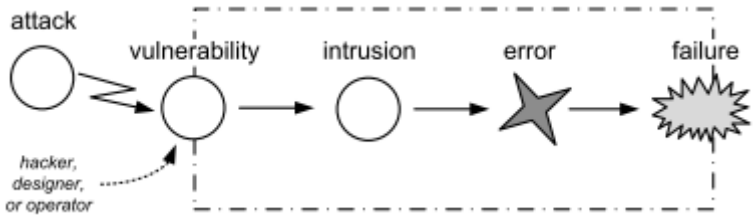

Figure 2 Intrusion Model

The terminology related to security attacks is defined as follows:

- **Attack**: a malicious interaction fault respective an intrusion attempt which aims at deliberately violating one or more security properties;
- **Vulnerability**: a fault deployed during operation or system design that can be exploited to create an intrusion;
- **Intrusion**: an externally-induced fault resulting from an attack that can be used to alter the system state.

An attack is an attempt to exploit a weakness or vulnerability in the system in order to perform an unauthorized action. A successful attack attempt results in an intrusion as illustrated in Figure 2 [2]. These vulnerabilities are in many cases functionalities of the system which are exploited by attackers while in rare cases they are mistakes or bugs which appear during the design phase of the attacked system itself.

The intrusion-tolerance paradigm as introduced in [3] assumes that systems remain to a certain extent vulnerable and that attacks on components or sub-systems will happen but only some of them will be successful. Therefore the goal is to ensure that the overall system remains secure and operational, with a measurable probability, even if some sub-systems are under attack or are actually failing. This is achieved by error processing mechanisms which ensure that a security failure is detected and/or prevented.

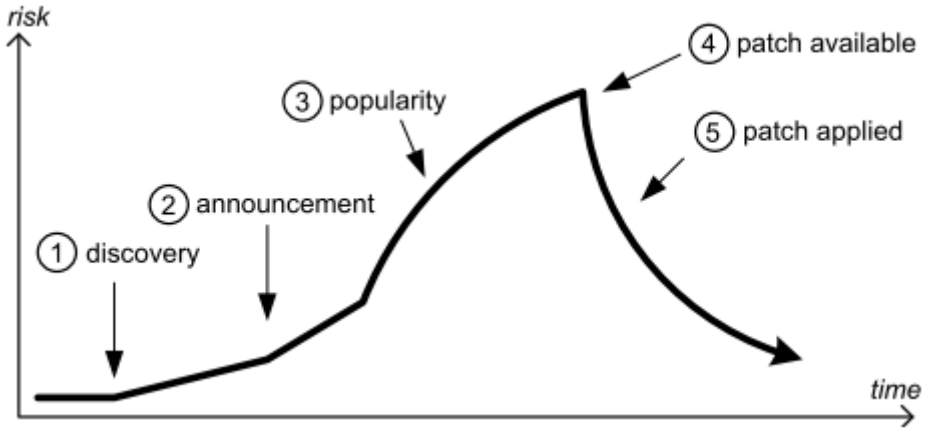

Figure 3 Vulnerability life cycle after Schneier

Schneier developed a vulnerability life cycle in [3]. Figure 3 presents this life cycle divided into five phases:

Phase 1: the vulnerability has not been discovered yet and it is dormant.

Phase 2: one or more people discovered the vulnerability and know how to make use of it, but no countermeasures exist. This phase poses the highest risk for a system, because nobody but the potential attacker knows about the vulnerability. At some point in time the discovered vulnerability will be made public (in phase 3).

Phase 3: By some means (bug report, scientific publication, exploit code, etc.) more people get to know about the exploit and the attack gains popularity.

Phase 4: automatic attack tools are available and reduce the technical skills to launch an attack. Now it should be fairly easy for many people to execute an attack and therefore the attacks number rises.

Phase 5: Finally, a patch is available and distributed to the users forcing the attack rate to go down.

Considering the above stated, the attack type needs to be defined. There are three types of possible attacks:

- "**Insider attack**" – these attacks use information from former employees and/or leaked information. The protection against these attacks is minimal as in most of the cases the security method applied is fully compromised;
- "**Lunchtime attacks**" – the attacker learns about the system and is able to use a narrow time window to execute his attack;
- "**Focused attack**" – the attacker invests a lot of resources and time in order to execute an attack.

Usually attacks have various motivations behind but these can be categorized as follows [4]:

- **Competition** (e.g. cloning, counterfeiting) – assumes knowledge and/or product theft in order to get a competitive advantage which allows a better market penetration;
- **Theft-of-Service** – obtaining access to an otherwise expensive or blocked-out service;
- **User authentication** (e.g. spoofing) – forging an users' identity to gain access to a system;
- **Privilege escalation** (e.g. feature locking) – gain unattained control over a system.

Attacks can be either local, where the attacker is considered to have physical access to the attacked system, or remote, where the attacker uses one or more communication interfaces to gain access to the attacked system.

During the product's life cycle, as depicted in Figure 4 [5], the product itself goes through a number of changes. The most important factor is market penetration, where the product is deployed and reaches numerous individuals. As the new product reaches the market, most probably it will not be attacked right at the beginning. After the first vulnerability is discovered, there is a certain timeframe until the information becomes widespread. During this time other attackers develop tools for easing the attack scenario and allowing for simple "push button solutions". Also during this phase, the developers become aware of the attack and begin to implement countermeasures. After this phase the attack becomes widespread as tools for executing the attack become available and more and more devices are compromised. This is the most critical period in time.

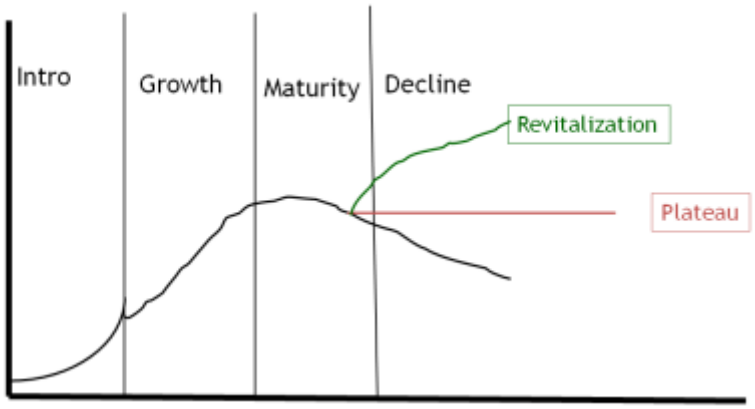

Figure 4 Product life cycle

This entire "flow" of a security attack needs to be considered while designing the product. Designers and developers need to be aware of present day security attacks and countermeasures as well as design techniques and state of the art technologies. Using dedicated design processes security needs to be an integral part of the designed system. Also security needs to be treated like any other function – to be upgradable (if its software) and versatile enough to be able to defend against new attacks (hardware especially as it's in the field, in case of industrial applications, for more than 10 years).

4. Requirements

The Internet enables any-to-any connectivity. The first wave of connectivity was user buildings (homes and offices) connecting to business buildings with wired Internet connections. The second wave was mobile user devices (laptops, smart-phones, tablets) connecting to businesses and each other over wireless Internet. The latest wave is “things” connecting to users, businesses and other “things” using mixtures of wired and wireless connectivity [6]. In order for this network of physical objects accessed through the Internet to properly work some properties are required: Trust, Security, Privacy, and Reliability.

4.1. Requirements

Table 1 Security Requirements

Requirement Type	Origin	Priority	Category	Description	Rationale	Fit Criterion	Dependencies	Conflicts
functional	WP3	MUST	Security & Privacy	communication shall take place over standard interfaces (e.g. I2C or SPI for Sensors and Ethernet between devices)	Using standard communication interfaces minimizes the development overhead and maximizes the code reuse			
security	WP3	MUST	Security & Privacy	data must be checked for functional correctness (e.g. identify defect and/or disconnected sensors and/or devices)	Transmitted data has to be valid in order to conserve bandwidth and assure the integrity of the entire system		HW/SW security module	longer processing times

Requirement Type	Origin	Priority	Category	Description	Rationale	Fit Criterion	Dependencies	Conflicts
security	WP3	MUST	Security & Privacy	data must be "secured" in order to allow a high enough security level: - eavesdropping -> encryption - data modification -> Encryption + integrity checks - replay attacks -> integrity checks - identity theft -> encryption + authentication - non-repudiation -> digital signature + encryption + authentication	only secured data can be trusted - plain text information can be modified while "traveling" over the Internet		HW/SW security module	longer processing times
security	WP3	MUST	Security & Privacy	secure storage for the on-device secret information (e.g. encryption keys)	a secure storage element is needed in order to provide a root of trust for the hardware secure boards		HW security module	system complexity; dedicated SoC structure

Requirement Type	Origin	Priority	Category	Description	Rationale	Fit Criterion	Dependencies	Conflicts
functional	WP3	MUST	Security	secure boot in order to have the device, every time, in a safe and known state	the system needs to execute only trusted software and run into a known state - for this very reason a secure boot mechanism is essential		HW security module	system complexity processing speed
functional	WP3	MUST	Security	secure update mechanism (e.g. update each device on its own)	secure update provides the means to upgrading the system		HW security module	system complexity; dedicated SoC structure
security	WP3	MUST	Security	secure enrollment mechanism (e.g. enroll each device in the system; if one device fails it will be automatically disabled)	each device needs to be uniquely identifiable and addressable		HW/SW security module	system complexity; dedicated SoC structure longer processing times
functional	WP3	MUST	Security	remote configuration	all VE should be remotely configurable		HW/SW security module configuration interface	

Requirement Type	Origin	Priority	Category	Description	Rationale	Fit Criterion	Dependencies	Conflicts
security	WP3	SHOULD	Security & Privacy	hardware root of trust (e.g. let the software rely on a secure element rather than make it secure on its own)	each VE with a hardware security board should be able to use the hardware security features as a root of trust - software should only handle the high level security operations		HW/SW security module	system complexity; dedicated SoC structure longer processing times
security	WP3	SHOULD	Security & Privacy	secure execution environment (e.g. split the execution environment into secure - where the core apps are running, and unsecure - where the non-vital apps, which require more processing time and are not system critical, are running)	secure VEs should have a clear separation between security & privacy critical apps and "the rest"		HW/SW security module	system complexity; dedicated SoC structure longer processing times

Requirement Type	Origin	Priority	Category	Description	Rationale	Fit Criterion	Dependencies	Conflicts
security	WP3	SHOULD	Security & Privacy	allow high level applications to use core hardware security features (e.g. remote configuration authentication performed using the secure element -> the software just triggers the element and the security part is handled in hardware)	secure VEs should be able to use directly hardware security functions whereas software only handles small parts of the communication & configuration -> HW root of trust		HW/SW security module	system complexity software support
functional	WP3	MUST	Security	use a standard OS which is verified and trusted (e.g. Linux)	standard OSes provide the necessary infrastructure, are verified and can be used "free of charge" (e.g. Linux)			
functional	WP3	MUST	Security & Privacy	use a secure server backend for key and data storage as well as for device enrollment	backend infrastructure is needed (e.g. Keystone)		SW security module on backend HW/SW security module frontend	software support

Requirement Type	Origin	Priority	Category	Description	Rationale	Fit Criterion	Dependencies	Conflicts
functional	WP3	MUST	Security	make the security "stuff" mostly transparent to the end user	security should "just be there" - users should not care about the infrastructure but rather use it			software support
functional	WP3	MUST	Security & Privacy	a unified API should spread over all VEs	all VEs should have the same API and signal via a flag which security level is provided		HW/SW security module	software support
functional	WP3	SHOULD	Security	There should be a mechanism which enforces authentication and access control to the cloud storage.	This is necessary to protect the large amounts of data that will persist in cloud storage.			

Requirement Type	Origin	Priority	Category	Description	Rationale	Fit Criterion	Dependencies	Conflicts
functional	WP3	SHOULD	Security	There should be a mechanism which ensures that metadata search results only contain data that the relevant user has read access privileges for.	This is necessary to prevent leakage of information via metadata search to unauthorized users.			

4.2. Trust

It is important to define a Trust Model that provides data integrity and confidentiality, and endpoint authentication and non-repudiation between any two system-entities that interact with each other [7]. According to [8] the definition of confidentiality is: “preserving authorized restrictions on access and disclosure, including means for protecting privacy and proprietary information” and integrity is defined as: “guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity”. A loss of confidentiality is the unauthorized disclosure of information and a loss of integrity is the unauthorized modification or destruction of information. Also in [8] there are also defined the authenticity as “integrity of a message content and origin, eventually including other types of information, e.g., timestamp, location, etc” and non-repudiation as “availability and integrity of the identical of a participant in the communication.” The goal is it to provide irrefutable proof of an action in the system to a third party.

In the past few years, various security protocols and standards have been developed to secure communication. Although a functional security protocol can theoretically protect the privacy and confidentiality of data, it cannot assure the trusted of the particular systems that inter-communicate using protocols. Attacks against Internet of Things exploit vulnerabilities of a particular system implementation to achieve their goals, rather than attempting to break cryptographic algorithms and protocols. A security vulnerability is a flaw, unintentionally developed or deliberately included in a system, which could later be exploited to cause a loss of system confidentiality, integrity, or availability.[9]

Most software and hardware components used in Internet of Things are imported from various sources, and cannot be treated as either certified or trusted. In order to have a trustworthy system we must assure trust in all system components and their interactions. We could do this by localizing trust in small set of systems components and entrusting these components to enhance trusted computing in all system components and interactions. Using such an approach would allow the build-up of the so-called root-of-trust from smaller components thus mitigating the risk associated with security attacks – if one component is breached the trust pyramid is still assured by the remaining ones.

This introduces additional vulnerabilities besides inadvertent development flaws. Example of possible attacks scenarios are presented below:

1. **Monitoring and sensing** the entities connected to Internet: sensor faults are common to all physical systems, Internet of Things attacks can target sensors to compromise sensory information. However many fault-tolerant techniques and algorithms have been developed to cope with them. [9]
2. **Communication and networking:** Networks are often used to exchange real time data between sensors, computing subsystems and other “things” connected. Communication between different entities is vulnerable to various attacks such as: eavesdropping, denial of services, man-in-the-middle, data modification, identity stolen. For example, in the case of eavesdropping- communication is listened by an unauthorized third party or for example in the case of man-in-the-middle- an attacker can pose as somebody else by stealing a digital identity. There are numerous research efforts that have already addressed these secure communication issues and those can be used to assure trusted communication. [9] Cryptographic accelerators could be a solution. As a result an attacker is unable to read the content of the intercepted communication or is unable to pose as the victim.

3. **Processing and computing:** A number of embedded controllers process sensing data and compute feedback decisions. An embedded controller is a computational platform incorporating a mixture of software-based and hardware-based processing devices, storage elements, I/O peripherals, and communication devices interacting together. Processing devices include simple micro-controllers, single and multi-core processors, digital signal processors, ASICs, and FPGAs. All known vulnerabilities of embedded controllers/processing devices represent threats for Internet of Things. Examples of vulnerabilities are presented below.

4.3. Security

The Security reference model presented in [8] is made of three layers: the Service Security layer, the Communication Security layer and the Application Security layer.

4.3.1. Communication security

Communication security means preventing unauthorized interceptors from accessing telecommunications in an intelligible form, while still delivering content to the intended recipients [10]. Two things must be taken into account for a Communication Security Model: variety of the entities involved in the system (data, machine, sensors, RFID, and so on) and a balance between security features, bandwidth, power supply and processing capabilities.

In Fig. 1 is presented a communication security model in which the IoT device space is divided into two main categories: constrained networks (NTC) and unconstrained networks (NTU). The communication between two categories is realized through gateway in order to assure the security. On the edge between the domains of unconstrained and constrained devices, gateways have the role of adapting communication between the two domains [8].

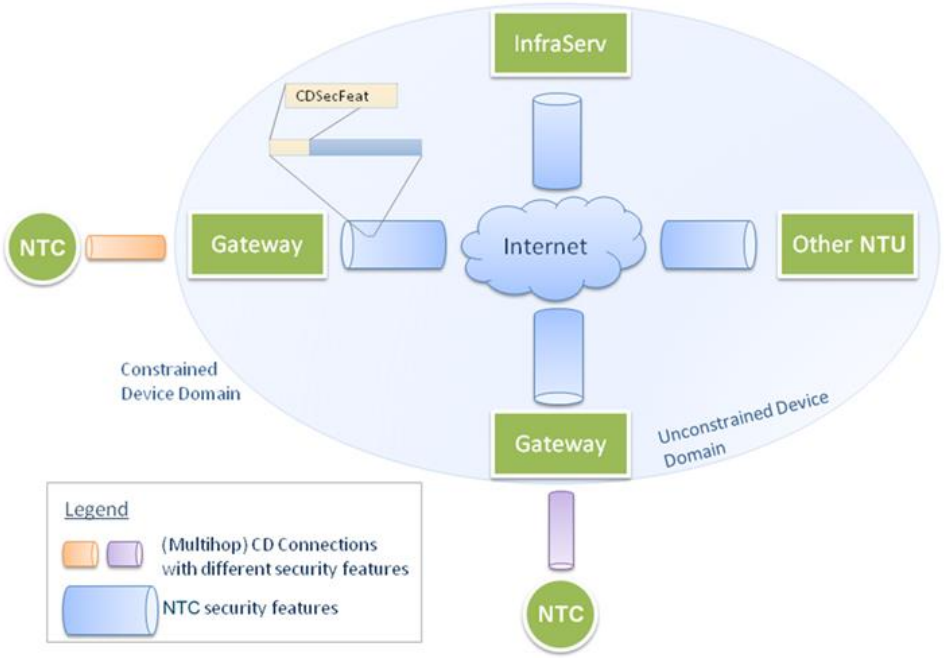

Figure 5 Communication Security Model

Features of the gateway designed to ensure security in communications are presented below [8]:

- Protocol adaptation between different networks;

- Tunnelling between themselves and other nodes of the NTU domain;
- Management of security features belonging to the peripheral network;
- Description of security options related to traffic originated from a node attached to the gateway;
- Filtering of incoming traffic according to network policies, user-defined policies, and destination-node preferences

4.3.2. Application security – safety layer

Most of the devices in the Internet of Things will be used in two broad areas:

- Critical Infrastructure – power production/generation/distribution, manufacturing, transportation
- Personal Infrastructure – personal medical devices, automobiles, home entertainment and device control, retail

Each of these uses must be reliable: a single failure in the system can lead to tragic consequences [19] – Risk Analysis, this is why it becomes important to assure also system safety. It is a common approach to achieve fail-safe systems comprising two phases: the identification phase- detecting all possible risks that could possibly lead to severe accidents and the system design according to the fail-safe philosophy.

4.4. Privacy

The domain of privacy partially overlaps security, including for instance concepts of appropriate use, as well as protection of information. Definition of privacy might be: the ability of an individual or group to seclude themselves or information about themselves and thereby express themselves selectively. In IoT a variety of entities are implied in handling user-generated data. As a consequence, the vast amount of user-generated data has led to growing concerns about privacy of its users. In Internet of Things the ability for entities to communicate in a secure environment is required, while at the same time preserving privacy. Privacy is all about control – enabling entities to maintain personal control over their personally identifiable information with respect to its collection, used and disclosure.

A privacy friendly system should guarantee the following properties:

- The subject must be able to choose sharing or not sharing information with someone else;
- The subject shall be able to decide for which purpose the information will be used as he is the right full owner;
- The subject shall be informed whenever information is used and by whom;
- During interactions between a subject and an IoT system, only strictly needed information shall be disclosed about the subject, and pseudonyms, secondary identity, or assertions;
- It shall not be possible to infer the subject's identity by aggregating/reasoning over information available at various sources;
- Information gained for a specific purpose shall not be used for another purpose - this also includes experience sharing.

Trust and privacy are considered as being two contradictory properties. From one side, we want that each entity be able to prove its own trust value and from the other side, we want that each entity does not disclose more personal information than it wants.

A solution to this problem is to calculate the trust value on the fly, based on certificates given to that entity in the many interactions it has had in the past. However, it has two major drawbacks: (1) The unique component would become a huge bottleneck in the system (2) It would become a single point of failure.

Another solution presented in [8] proposes that subjects are allowed just one trust-value, valid for a certain number of pseudo-entities, and included in a trust-certificate signed by the AuthN component.

4.5. Exploitable Features

As an IoT platform, COSMOS is formed by different components such as the IoT platform itself, running in one or more data centres or embedded device which feed data into COMOS. Data centres form the basis for every IoT platform and are not the object of the present project. Still, the embedded devices which produce and consume information need special care with respect to security. A system is as secure as its weakest link which in this case is caused by the large number of smart devices which are interconnected with COSMOS.

As with every system there are a number of traits which can be used to characterize the system. These traits are summarized in the tables below.

Table 2 Firmware Update

Attack description	Problems due to attacks (compromised asset)	Solution description	Result
The firmware update is intercepted and modified	Devices do not get the new firmware or get the wrong firmware update	Encryption of firmware updates Hashing of firmware updates	Wrong firmware images are rejected
An authorized firmware update is triggered	Devices get the wrong firmware update potentially containing malware	Encryption of firmware hashes for update verification Authentication of update issuer	Wrong firmware updates are rejected Wrong issuer is recognized and update is not performed

Table 3 Code Execution

Attack description	Problems due to attacks (compromised asset)	Solution description	Result
Code execution is altered by changing FLASH memory content	Firmware is altered and functionality is changed/added to the attackers desire	FLASH memory hashing Secure boot	FLASH memory changes (at start-up) are detected and the system will not power up

			in case of an attack
Code execution is altered by changing RAM memory content	At runtime variables are changed possibly changing execution flow	RAM hashing Secure Execution	System will detect RAM memory changes and react accordingly or will prevent changes at all
Communication packets are changed	Communication data is altered(e.g. metering data)		Attacker is unable to decrypt the data and thus to alter its content

Table 4 Storage

Attack description	Problems due to attacks (compromised asset)	Solution description	Result
An attacker tries to retrieve information from a memory	Data is stolen (PINs, digital IDs, sensitive data, etc)	Secure storage Authentication Encryption	Attacker is unable to read memory content Attacker's intrusion is detected

Table 5 Physical Intrusion

Attack description	Problems due to attacks (compromised asset)	Solution description	Result
Third-party components can contain logic bombs, worms, viruses, Trojan, trap doors	Allowing execution of illegitimate actions violating system operating and security policies with malicious objectives	Encryption Secure storage Secure execution Sensors	Attacker is unable to get physical access to the digital IC without being detected – the IC takes action upon attack detection

Table 6 Cloning

Attack description	Problems due to attacks (compromised asset)	Solution description	Result
An attacker tries to clone the software	Device functionality is cloned	Full memory encryption Memory obfuscation	Memory content cannot be read

These exploitable traits were identified after a security risk analysis, as presented in D2.3.1 [19].

Table 7 Identified Risks

No.	Risk	Risk Rating
1	Unavailability of IT infrastructure	HIGH
2	Loss of data due to unattended access to COSMOS	HIGH
3	Loss of data due to improper data handling	LOW
4	Loss of confidentiality due to improper user management	HIGH
5	Loss of confidentiality due to improper data handling	LOW
6	Loss of data integrity	MEDIUM
7	Unattended access to COSMOS which might compromise the system availability	MEDIUM
8	Un-trusted VEs might get access to COSMOS	MEDIUM

Each of these risks can be realised using many attack types as well as different efforts. Also these risks can be posed be one or more system components, as described above, therefore system-wide security measures need to be enforced. Even more, security needs to be an integral part of the system, not just an add-on.
For these very reasons, the COSMOS system architecture not only presents the functional components but also security features and non-functional components meant to raise the overall confidence level and build-up the trust pyramid.

5. Security Assessment and Standardization

This chapter comprises some of the most important standards needed for developing, testing and preparing security enabled systems for.

As security is a very delicate matter for all involved parties there are no golden rules to be applied but rather a series of guidelines and standards which ensure that a carefully designed system meets certain criteria allowing it to be secure. For this very reason “security standards” are to be seen as guidelines and/or a “code of practice”. There are a number of different organizations which seek provide either national or international standards.

5.1. Standards

5.1.1. International Organization for Standardization (ISO)

The International Organization for Standardization (ISO) is a consortium of national and international standards institutes and is by far the best known standardization institution. The relevant standards in information security are:

- ISO 15443: "Information technology - Security techniques - A framework for IT security assurance",
- ISO/IEC 27002: "Information technology - Security techniques - Code of practice for information security management",
- ISO-20000: "Information technology - Service management"
- ISO/IEC27001: "Information technology - Security techniques - Information security management systems - Requirements" are of particular interest to information security professionals.
- The ISO27000 family originates from the British Standards Institution BS7799 standard which first appeared in 1995. The second issue of the BS7799 standard appeared in 1999 and forms the basis for the ISO27000 family.
- The ISO27000 family is the most used information security standard and is applied around the world. Companies have to implement methods and measures specified in ISO27002 while audits are conducted based on ISO27001.

5.1.2. National Institute of Standards and Technology (NIST)

The National Institute of Standards and Technology is a non-regulatory federal agency within the Department of Commerce. Responsible for information security is the *Computer Security Division*. This division uses the Security Resource Center to develop standards, metrics, tests and validation programs for increasing the security awareness and strengthening security policies. The NIST covers planning, implementation, operation and management of security topics and publishes regularly the Federal Information Processing Standards (FIPS). Under the FIPS umbrella there are a number of relevant standards which have to be considered out of which the FIPS 140-2 “Security Requirements for Cryptographic Modules” is the most important for cryptographic applications and are serving as guidelines for the Hardware Security Board developed within COSMOS.

5.1.3. The Internet Society (ISOC)

The Internet Society is a non-profit organization which provides standards and security policies for the Internet. Besides having an important role in governing the Internet, the ISOC is responsible for releasing the Internet infrastructure standards. This is done by the Internet Engineering Task Force and the Internet Architecture Board which are hosted by the ISOC directly. The ISOC hosts the Requests for Comments (RFCs) which includes the Official Internet Protocol Standards and the RFC-2196 “Site Security Handbook”. Although this organization is focused only on the Internet it still has a far-reaching impact to all devices which are connected to a network with an Internet gateway. As today the Internet of Things paradigm is ever-evolving the RFCs become more and more relevant to embedded devices and their respective software applications.

5.1.4. Information Security Forum (ISF)

The Information Security Forum offers standards of good practice, methodologies, benchmarking tools and advice in form of conferences for its members. The 2012 Standard covers current information security 'hot topics' such as consumer devices, critical infrastructure, cybercrime attacks, office equipment, spreadsheets and databases and cloud computing. It is used to build a comprehensive and effective information security management system.

5.1.5. Bundesamt für Sicherheit in der Informationstechnik (BSI)

The IT Baseline Protection Catalogs are a collection of documents from the *German Bundesamt für Sicherheit in der Informationstechnik*. Among its publication, the “IT Baseline Protection Catalogues” is the most important one. The 2005 release, updated in 2007, covers all aspects of information security. With almost 3000 pages the German standard is by far the most comprehensive one available. It consists of four basic parts:

- “General Information” – defines security, introduces basics related to information security, defines roles and terms used throughout the standard;
- “Modules Catalogues” – presents "Generic Aspects of IT security";
- “Threats Catalogues” – details aspects of various security threats;
- “Safeguard Catalogues” – presents methods for protecting and safeguarding sensitive data.

Although there are a number of security standards, these cover mostly information security and do not provide certification criteria for hardware based security devices. Still two of the main certification organizations are the Common Criteria and the Communications-Electronics Security Group.

5.2. Assessment and Certification

5.2.1. Common Criteria (CC)

The “Common Criteria” is an international undertaking targeting security evaluation. It is based on previous evaluation schemes and is built upon the expertise of governmental and institutions dedicated to security.

There are two possible evaluations: for products and for protection profiles. A protection profile is an implementation-independent set of security requirements for a category of products or systems that meet specific consumer needs. It provides a through description of threats, environmental issues and assumptions, security objectives, and Common Criteria requirements for a family of products.

In order to evaluate a single product, a so called “security target” has to be either derived from a protection profile or developed on its own. This is a set of requirements and specifications for a particular product.

The seven “Evaluation Assurance Levels” or short “EAL” are:

- EAL 1: “Functionally Tested Analysis” of security functions based on functional and interface specifications. It is applicable to systems where security threats are not serious.
- EAL 2: “Structurally Tested Analysis” of security functions including the high level design. Evidence of developer testing based on functional and interface specifications, independent confirmation of developer test results, strength-of-functions analysis, and a vulnerability search for obvious flaws must be provided.
- EAL 3: “Methodically Tested and Checked” is basically the same evaluation criteria as in EAL2 which additions referring to the use of development environment controls and configuration management. This level provides a moderate level of security.
- EAL 4: “Methodically Designed, Tested, and Reviewed”. This level requires a low-level design, complete interface description, and a subset of the implementation for the security function analysis. Additionally, an informal model of the product or system security policy is required. This level targets systems with a moderate to high security requirement. Examples of EAL4 certified products are Microsoft Windows Server, commercial Linux server editions from companies like Red Hat or Novell.
- EAL 5: “Semiformally Designed and Tested”. A formal model, a semi formal functional specification, a semi formal high-level design, and a semi formal correspondence among the different levels of specification are required. This level is applicable for smart cards (e.g. Infineon SLExx family) and multilevel secure devices.
- EAL 6: “Semiformally Verified Design and Tested”. This level builds upon EAL5 with the added requirements for semi formal low-level design and structured presentation of the implementation.
- EAL 7: “Formally Verified Design and Tested” is the highest level of evaluation. It requires a formal representation of the functional specification and a high-level design, and formal and semi formal demonstrations must be used in correspondence.

5.2.2. Communications-Electronics Security Group (CESG)

The Communications-Electronics Security Group is part of the The Government Communications Headquarters, a British intelligence agency responsible with information assurance and signal integrity to the armed forces. CESG is the UK national technical authority for information assurance, including cryptography. CESG does not manufacture security equipment, but works with industry to ensure the availability of suitable products and services.

The CESG has developed alternative schemes, covered under the Service Catalogue, to the Common Criteria in order to have better control over the certification process and reduce the overhead caused by the international setup of the Common Criteria.

The BSI reports as well as the Common Criteria guidelines, together with the Architectural Reference Model in IoT [18] have served as guidelines for the WP3 architectural driven activities.

6. Architecture

As presented in WP2, the COSMOS architecture has many individual components which are linked together by the “data bus”. This internal structure requires two external “partners” – the data generators and data consumers, where non-human entities are called VEs. As depicted in figure 6, the COSMOS architecture, as defined in WP2, form the COSMOS platform – the COSMOS services and the “data bus” are sub-components of the platform itself. If COSMOS is viewed as a black box, there are just two interaction points, one feeding data into COSMOS (VEs) and the other retrieving data from COSMOS (e.g. humans of machines - VEs).

Taking this into account as well as the previously identified risks and exploitable features, the COSMOS platform is enhanced with security features as described below.

The COSMOS platform is protected with the help of a gateway from the outside world. This gateway provides security based authentication as well as tractability for both VEs and end-users.

Security functions are managed by the Security Management Module (SMM) which is part of the COSMOS platform, like all other COSMOS services. The SMM provides key generation and management services as well as user and security management functions.

From the functional point of view, VEs and end-users have different APIs at their disposal, all of which enable them to use the various COSMOS services. In order to provide the required level of security, a unified security API is used for all COSMOS partners. This enables COSMOS to keep an overview over the security features as well as to be able to manage, using a unified approach, all partners.

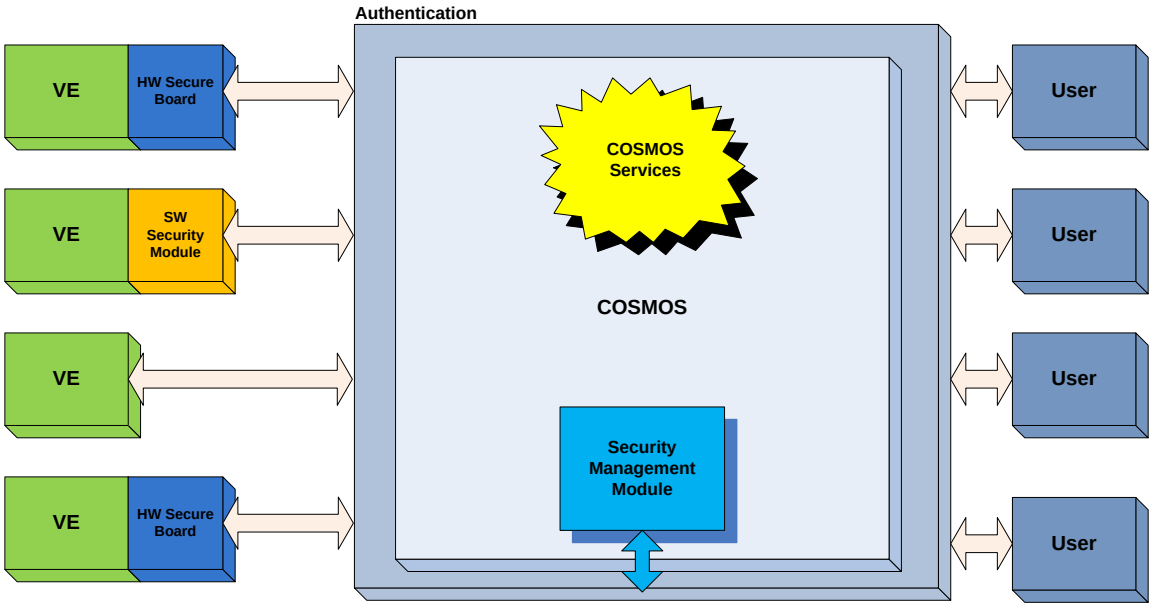

Figure 6: COSMOS Security Architecture

COSMOS, as a platform, has interactions with two partners: VEs and end-users, which are managed by the SMM.

Virtual Entities

Virtual Entities are interacting with the COSMOS platform either as data generators or consumers. There are three classes of Virtual Entities:

- High-secured (Hardware Security Board)
- Low-secured (SW Security Module)
- Non-secured (No Security module)

Using the unified security API, a read-only flag symbolizes the security level of each VE. Based on this each VE is assigned a rating which can either be kept or downgraded, if the VE performs poorly (e.g. many unknown packets sent out, drop-rate is high, etc.).

From the basic API level point of view as well as from the security requirements, there is no difference between VEs which generate and which consume data, as well as between machines (e.g. embedded IoT devices) and humans. The only difference is presented by the different methods of interaction between the COSMOS platform and the entities.

Initially, Virtual initially do not have any encryption key. As the entire security concept of COSMOS relies on unique key based authentication and strong encryption, each VE needs to have a private key. Therefore each entity which interacts with COSMOS needs to have a unique key, generated within COSMOS, and distributed to the respective entity using a communication method available or by storing it on-chip (e.g. for the Hardware Security Board). Key distribution mechanisms rely on Diffie-Hellmann algorithm using any communication interface available as a carrier (e.g. SSH or HTTPS) while humans can receive their keys per email.

There are two possible ways to register a new VE. The first possibility is that after its first "Power-On", it protocols with the Security Management Module and based on the unique MAC address of the Hardware Security Board (which is recognized by COSMOS) it receives a unique cryptographic key – or using a pre-programmed software ID. The second possibility is similar to the one above described just that instead of a MAC address, the key exchange is performed based on an initial key securely stored on-chip – this method only applies to high-secured VEs.

Should a new VE be enrolled in COSMOS, first it has to get its cryptographic key. This operation can be performed either via email (for humans) or through SSH protocol (the enrolment procedure through SSH uses a generic key and is both available for humans and machines). The key exchange mechanism has to be accomplished over a SSH connection. With the new generated key, the Virtual Entity and the user are able to authenticate in the system.

Security Management Module

The COSMOS environment can be viewed as a black box which provides various services to VEs. These services handle three basic data types:

- Security critical – information is both secret and privacy critical;
- Security aware – information which can be secret but is not privacy critical;
- Non-secure – public information which contains no secret and is not privacy aware.

As a black box, COSMOS needs to handle the three basic data types, thus it needs to provide following services:

- Authentication – while VEs need to be authenticated into COSMOS, data has to be genuine. In this context, both communication parties need to validate each other in a consistent manner;
- Integrity – authenticated data has to be accurate and consistent over its life cycle, from source to destination;
- Non-repudiation – none of the parties should be able to deny its actions within COSMOS;

- Availability – the information needs to be accessible when required and with minimal delay.

Therefore, the security management module serves as a generic security gateway for the COSMOS environment. Following an iterative design approach, the Security Management Module provides basic security mechanisms which strengthen the COSMOS environment.

The goal of the Security Management Module is to allow only authenticated VEs access to the COSMOS environment thus enabling the COSMOS services, running within the COSMOS environment, to trust the information.

7. Security components

7.1. Hardware coded security

The Hardware Security Board consists of a physical hardware device which provides the link between sensors (data generators) and the COSMOS environment/platform. The Hardware Security Board can be either attached to one sensor or can be a hub for an entire collection of sensors (e.g. temperature, pressure, humidity, surveillance cameras, etc.).

In order to provide high trustworthiness a hardware coded security forms the foundation of the HW Security Board. This layer provides basic security primitives which are used by software drivers and applications as the so-called “root of trust”.

The Hardware Security Board consists therefore of a FPGA platform device (e.g. Xilinx Zynq [17]). This provides the necessary means for developing the hardware coded security components while making use of standard, state-of-the-art computing processors. The operating system of choice is Linux which provides not only the necessary platform for developing the high-level software applications but also enables the usage of the security hardware modules.

The hardware components within the Hardware Security Board provide:

- Secure Boot – using encrypted flash memories and device-unique keys, enables only trusted software applications to be executed;
- Secure Storage – allows for on-chip key storage while protecting against common security attacks which target key recovery;
- Secure execution – using hardware partitioning schemes, unsecure software applications are sandboxed, thus protecting the rest of the computing platform from malicious software or malware;
- Cryptographic hardware accelerators – allow for fast, on the fly encryptions and decryptions to be performed, without performance loss.

7.1.1. Hardware Security Board

In the COSMOS context, the demo HW Security Board will be implemented on a Field Programmable Gate Array (FPGA) based platform. The proposed solution has multiple advantages:

- Design flexibility
- High performance
- Fast turnaround time
- High resources density (internal RAM blocks, GPIOs, DSPs, etc)
- Low cost (for one prototype)

Having all these facts in mind, after the selection process it was chosen the ZC702 evaluation board from XILINX. The features of this general purpose evaluation board are listed below:

- Zynq-7000 XC7Z020-1CLG484C AP SoC (containing 2 x ARM Cortex A9 Application Processors)
- 1 GB DDR3 component memory (four 256 Mb x 8 devices)
- 128 Mb Quad SPI flash memory

- USB 2.0 ULPI (UTMI+ low pin interface) transceiver
- Secure Digital (SD) connector
- 3 x Clock sources
- Ethernet PHY RGMII interface with RJ-45 connector
- USB-to-UART bridge
- I^2C bus
- Status LEDs
- User I/Os (FPGA Mezzanine Card (FMC) Interface → can be used for board extension modules)
- Dual 12-bit 1 MSPS XADC analog-to-digital front end

The ZC702 block diagram is shown below.

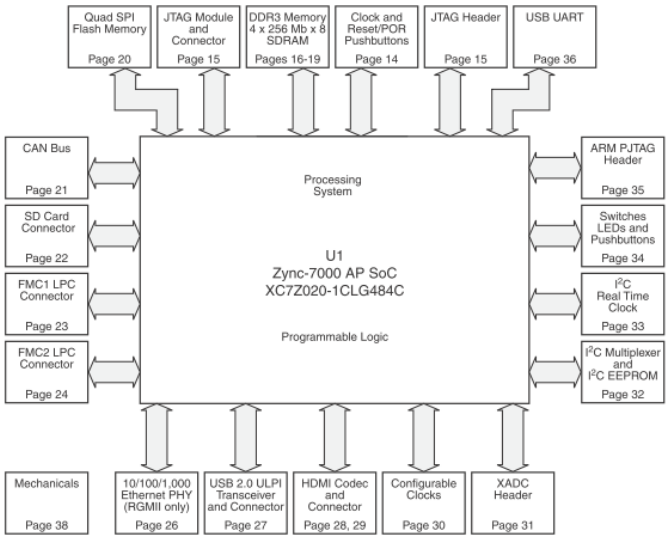

Figure 7: ZC702 Board Block Diagram

The ZC702 board is populated with the Zynq-7000 XC7Z020-1CLG484C AP SoC. The XC7Z020 AP SoC consists of a SoC-style integrated processing system (PS) and programmable logic (PL) on a single die. The PS integrates two ARM® Cortex™-A9 MPCore™ application processors, AMBA® interconnect, internal memories, external memory interfaces, and peripherals including USB, Ethernet, SPI, SD/SDIO, I2C, CAN, UART, and GPIO. The PS runs independently of the PL and boots at power-up or reset.

There are several possibilities to configure the platform: from QSPI flash memory, from SD card, USB JTAG and Platform cable header. The most “secured” way to bring the platform up is to configure it from an encrypted SD content.

The 1 GB, 32-bit wide DDR3 memory system is comprised of four 256 Mb x 8 SDRAMs (Micron MT41J256M8HX-15E). In normal operation mode, the application can frequently use this external memory.

The Quad-SPI flash memory located at provides 128 Mb of non-volatile storage that can be used for configuration and data storage. Since this is also an external memory, if the application uses it, the communication between FPGA and QSPI flash has to be encrypted.

The ZC702 board uses the Marvell Alaska PHY device for Ethernet communications at 10 Mb/s, 100 Mb/s, or 1,000 Mb/s. The board supports RGMII mode only. The PHY connection to a user-provided Ethernet cable is through a Halo HFJ11-1G01E RJ-45 connector with built-in magnetics. On power-up, or on reset, the PHY is configured to operate in RGMII mode.

The ZC702 board implements a single I2C port on the XC7Z020 AP SoC (IIC_SDA_MAIN, IIC_SDA_SCL), which is routed through a TI Semiconductor PCA9548 1-to-8 channel I2C bus switch. The bus switch can operate at speeds up to 400 kHz. The sensors can be connected to the HW platform through this bus. However, if other interfaces are needed in order to connect different sensor types to the HW platform, these interface modules can be easily designed and integrated into the FPGA and the corresponding GPIOs can be driven through the FMC connector to the extension board(s).

The ZC702 board supports the VITA 57.1 FPGA Mezzanine Card (FMC) specification by providing subset implementations of low pin count (LPC) connectors. Both connectors use a 10 x 40 form factor that is partially populated with 160 pins.

The ZC702 board provides an Analog Front End XADC block. The XADC block includes a dual 12-bit, 1 MSPS Analog-to-Digital Convertor (ADC) and on-chip sensors. This Analog to Digital Converter can be used for internal/external analog input measurements. For example, it can be used to monitor the internal die temperature, internal voltages as well as any external analog input.

The FPGA platform supports virtually any type of sensor with SPI/I2C/analog interface. Also, due to the high number of available GPIOs and to the possibility to connect extension boards, a high number of sensors can be connected to the FPGA platform.

For highly secured applications, where the communication channels between HW Security Board and all its connected sensors has to be secured, it can be used an “intelligent sensor” which is compound of a normal SPI/I2C sensor, tightly connected to a small microcontroller both encased in resin so that if a potential hacker tries to decapsulate the chip, it will permanently damage the encapsulated system. The microcontroller from the “intelligent sensor”, which can be addressable (8b/16b/32b address) over the I2C protocol, collects the data from sensor encrypts it and sends it to the HW Security board. If encryption methods are used, the device-unique encryption key can be periodically updated through the security board based on Diffie-Hellman key exchange algorithm.

7.1.2. Diffie-Hellman key exchange

The Diffie-Hellman key exchange mechanism is a specific method of exchanging cryptographic keys. This method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can be used to encrypt/decrypt subsequent communications using a symmetric key-cipher. Here is a short example of how Diffie-Hellman mechanism is working:

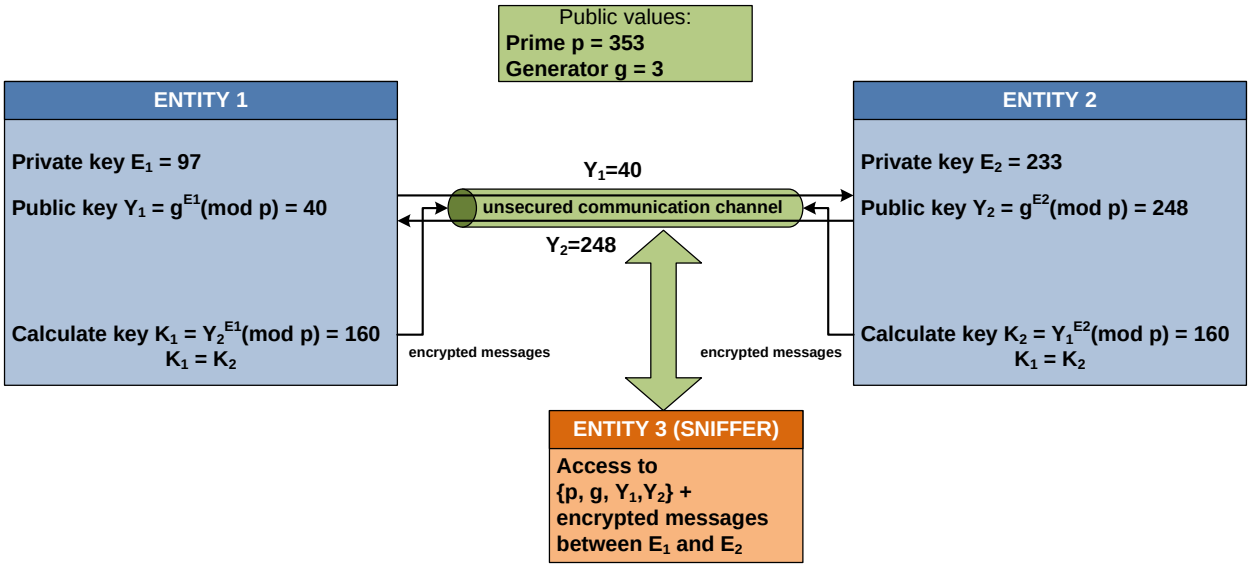

Figure 8: Diffie-Hellman Key Exchange Algorithm

Let us assume two entities that want to communicate to each other over an unsecured link (e.g. Ethernet protocol). The idea of Diffie-Hellman algorithm is that it's easy to compute powers modulo a prime number but hard to reverse the process. If the prime number is large enough, e.g. 1024 bits, a brute-force attack is virtually impossible. The algorithm steps, which will be implemented in all SMM, with special focus on hardware, are:

- Entities 1 and 2, using an unsecured communication link, agree on a prime number *p* (353) and a generator *g* (3).
- Entity 1 selects an integer number $E_1<p$ (97). On the other side, Entity 2 chooses an integer $E_2<p$ (233). Both numbers have to be kept secret because these are their private keys.
- Entity 1 calculates its public key $Y_1=g^{E1}(mod\ p)=3^{97}(mod\ 353)=40$ and sends it to the communication partner. Likewise Entity 2 computes its public key $Y_2=g^{E2}(mod\ p)=3^{233}(mod\ 353)=248$ *and sends it to Entity 1.* Sending these 2 public keys over an unsecured channel is safe because it is very difficult for a third party (Entity 3) to calculate E_1 from Y_1 and E_2 from Y_2.
- After the public key reception, both entities will be calculating the shared private key which can be used to encrypt/decrypt their communication.
    - $K_1=Y_2^{E1}(mod\ p)=248^{97}(mod\ 353)=160=K_2$
    - $K_2=Y_1^{E2}(mod\ p)=40^{233}(mod\ 353)=160=K_1$

The conclusion is that Entity 3 cannot easily compute the private keys of the other two entities only by sniffing the communication channel if the public prime number is large enough. However, if this entity intercepts all packages sent between Entities 1 and 2, it can act like Entity1 for Entity2 and vice-versa and sends its own public keys to both communication partners. By faking the link between the communication partners, it practically breaks the encryption.

This kind of data packets interception can be counteracted with the introduction of hash functions. A hash function is any algorithm that maps data of arbitrary length to data of a fixed length. The values returned by a hash function are called hash values, hash codes, hash sums, checksums or simply hashes. Hash functions are primarily used to generate fixed-length output data that acts as a shortened reference to the original data. This is useful when the original data is too cumbersome to use in its entirety. Hash functions are typically not invertible, meaning that it is not possible to reconstruct the input datum x from its hash value h(x) alone. In many applications, it is common that several values hash to the same value, a condition

called a hash collision. Since collisions cause "confusion" of objects, which can make exact hash-based algorithm slower and approximate ones less precise, hash functions are designed to minimize the probability of collisions. For cryptographic uses, hash functions are engineered in such a way that is impossible to reconstruct any input from the hash alone without expending great amounts of computing time.

In the above presented case, any altering attempt of Entity 3 to a specific encrypted data packet between the other 2 entities will cause packet abortion and alarm flag rise.

7.2. Cloud storage security

There are many important security aspects related to cloud storage. We focus here on security aspects of the cloud storage components developed for COSMOS, specifically metadata search and analytics close to the storage.

One important aspect of cloud storage security is access control – ensuring that only those with access rights to the data can access it or change it. Typically, data storage systems have mechanisms which enforce access control on their data. This is also true for OpenStack Swift [11], which is used to implement the Data Store component of COSMOS. Swift data is stored as objects organized into containers. Swift allows defining Access Control Lists (ACLs) at the Swift container level, which state which users can read and/or write objects in a container, as well as which users can list the objects in a container.

However, correct enforcement of access control becomes more complex when object storage such as Swift is enhanced with new capabilities. For example, by using metadata search, a user may be able to access information that she was not able to access using the object storage alone. As another example, by using storlets, a user may be able to delete objects that it didn't previously have write permissions for. In this document, we describe a design for metadata search whose results contain only resources that the user is authorized to access. Deliverable 4.1.1 describes sandboxing techniques for storlets, which ensure that computations can only access data it is supposed to access.

We adopt Keystone as a framework for authentication and authorization with Swift, and adopt the Keystone middleware for Swift. Its position in the middleware pipeline should precede the middleware for metadata search. We propose the following approach for metadata indexing and search.

Indexing

Any write requests (PUT/POST/DELETE) which are unauthorized will not reach the index, because of the positioning of the middleware pipeline.

Search

A metadata search is a Swift GET request with a particular metadata search header. The URL of this GET request denotes which containers and objects are referenced by the search. There are 2 types of searches: searches within a container and cross container searches.

Searches within a container should be only being granted if the user has permissions to both list objects in the container and to read the objects in the container. The metadata search middleware should allow the search if it can verify that these permissions are granted.

Cross container searches can potentially access all containers in a Swift account. Therefore we propose granting permissions for such a search only when the user has the keystone Account Owner role. This should be enforced by the metadata search middleware.

7.3. Privacy

Within the context of COSMOS virtual entities will be able to contact each other as well as with the platform. This information exchange has a high possibility of violating the user's privacy in a way that he is not aware of.

Our goal is to ensure that every VE (and therefore its user) shares only the intended information and leaves out all other that is considered as private and is believed to affect its privacy.

The design and implementation of privelets follow an iterative approach. As a first step, messages about to be sent from a VE are analysed and specific fields are filtered based on specific configuration options.

A privelet is a mechanism embedded (or external) to the VE which acts as a filter.

It lets all the public fields to be exchanged and it keeps safe all the private ones.

Data generated from the device must pass through that filter and then be exported to another VE or our COSMOS platform. That's a privelet's action. The VE generates data with a certain interval update and wants to share it through our platform or between another VE. It generates a JSON file where all those data pass through this mechanism called privelet. The VE configures the privelet by setting values to the data exchanged and thus a configuration file is stored in the privelet. After that any data exchanged from the VE, will have the same configuration regarding the privacy level of each tag in order to decide what data to send to the platform and what to keep. The privelet will not change each tag's privacy level unless the VE calls it again and re-configures it. After this configuration the data exchanged will be filtered and exported with the privelet's configuration.

During the first time of the VE's operation, it can choose the privacy level of any tag exchanged by the VE between "private" and "public" and configure them.

By filtering all those considered as private tags we could eliminate the chances of de-identification caused by quasi-identifiers. So if someone would like to choose all his tags to be sent on the platform, it is highly possible that anyone could use databases that store other information regarding his personal life and are linked to the data exchanged though COSMOS platform and build a map about him being able to track or predict the further moves on his life. By using the privelets, we give the user the ability to avoid all this and in this way we expect to make data transferring and exchanging anonymous and ensure privacy for our users.

In order to design and implement the privelet mechanism we use RESTful services [12] and more specifically the JAX-RS API. Jersey RESTful Web Services [13] framework is open source. We use it in order to build the Web services in Java because it provides excellent support for the JAX-RS API. A lightweight server will be used to host the implemented services.

Assuming that data is transferred through JSON files, another reason to pick Jersey is the fact that Jersey has a set of extension modules, where each of these contains an implementation of a feature to support JSON files.We picked MOXy [14] . JSON binding support via MOXy is a default way to support our JSON binding in our application which will be implemented with Jersey. In that way we implement and design the @GET and @PUT HTML methods used in our code. The payload of each request has the useful information in JSON format.

Another means to configure privelets will be developed, which will allow a human user to configure the privelet via a User Interface. This will be based on JSF 2.2 combined with primefaces 4.0. JavaServer Faces (JSF) [15] is a Java specification for building component-based user interfaces for web applications while PrimeFaces [16] is a lightweight library. The UI will

work as follows. When the privelet receives a message containing data of an unkown type it will postpone the sending of the message and provide the human user with an analysis of the data to be sent, prompting for selection of which fields are private and which are not. The selection updates the configuration of the privelet.

8. Conclusions

This report “End-to-End Security and Privacy (Design and Open Specification)” presents the results of the work carried out in the scope of WP3 “End-to-End Security and Privacy” of the COSMOS project during the first 8 project months.

The first step was to identify the functional components of COSMOS which need to be secured. In addition to the WP2 developed architectural components, using the Common Criteria, IoT-A Reference Architecture and BSI Baseline Security Catalogues, a thorough security risk analysis was performed. The goal was on the one hand side to identify and mitigate security risks by enhancing the COSMOS platform and on the other hand side to improve design efficiency and build upon the vast experience already available.

Furthermore an analysis of the input from WP2 tasks 2.1- Market Analysis and 2.2- Requirements as well as task 2.3 – Architecture Specification was used as input for WP3 and the present report. WP3s tasks are split such that they cover all aspects of COSMOS – device level security, cloud level security and data flow level security. Using a common set of guidelines and tools, WP3s tasks work together towards a unified security architecture enforced throughout the COSMOS platform.

Using a top-down approach, the COSMOS Security Architecture is split into sub-components which are described in detail. As the project evolves, further gaps will be filled out by enhancing the architecture with new components. Therefore a modular approach was used in order to facilitate the extension and mitigate redesign efforts. Security, performance and ease of use are main drivers in new activities. Interactions between components, interfaces and processes will be described extensively as detailed specification for each component will be available. The Security Management Module, which serves as a central hub for security enabled activities will be extended with new functionality as the project architecture crystallizes. WP3 will further rely on WP2 and the specified use-cases to developing guidelines and security enabled APIs for the project. It is expected that this report will be useful input for the whole project and especially for the development WPs (WP4, WP5 and WP6) that are the main consumers of the work carried out in WP3.

WP3 is in continuous collaboration with the other WPs of the project, especially with WP2, in order to acquire more requirements and to “fine-grain” the architecture design. WP3 will use feedback from the development work packages in order to “fine-tune” the architectural approach and to ease the other WPs activities.

9. References

[1] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing, 1(1):11–33, January-March 2004.

[2] David Powell and Robert Stroud. Conceptual model and architecture of MAFTIA. MAFTIA deliverable D21, January 2003.

[3] Bruce Schneier. Full disclosure and the window of exposure. Crypto-Gram Newsletter, Online, September 2000.

[4] Understanding hardware security, Grand, J., Grand Idea Studio Inc., Black Hat Conference Proceedings, Japan 2004.

[5] Consumer prihologist, retreieved aprl.2014, online available at: www.consumerpsychologist.com.

[6] Security the Internet of Things, http://www.sans.org/event/internet-of-things-summit

[7] Internet of Things-Architecture, Delivarable D1.5 Final architectural reference model for the IoT v3.0

[8] FIPS PUB-199, feb 2004, from Standards for Security Categorization of Federal Information and Information Systems homepage, retrieved oct 2012

[9] Mohammed M. Farag, "Architectural Enhancements to Increase Trust in Cyber-Physical Systems Containing Untrusted Software and Hardware", Dissertation submitted to the Faculty of the Virginia Polytechnic Institute and State University

[10] Wikipedia, Communications Security

[11] OpenStack Object Storage API v1 Reference http://docs.openstack.org/api/openstack-object-storage/1.0/content/index.html

[12] RESTful: http://en.wikipedia.org/wiki/Representational_state_transfer

[13] Jersey: https://jersey.java.net/

[14] MOXy: https://jersey.java.net/documentation/2.7/user-guide.html#json.moxy

[15] JSF: http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

[16] Primefaces: http://www.primefaces.org/whyprimefaces

[17] Xiliniz Zynq. Retrieved aprl. 2014. Available online at : http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/.

[18] F. Carrez *et al.*, "IoT-A Deliverable D1.5 – Final Architectural Reference Model for the IoT v3.0", www.iot-a.eu/public/public-documents/

[19] COSMOS Project D2.3.1 Conceptual Model and Reference Architecture.