COSMOS

Cultivate resilient smart Objects for Sustainable city applicatiOnS

Grant Agreement Nº 609043

D3.1.3 End-to-End Security and Privacy: Design and Open Specification (Final)

WP3 End-to-End Security and Privacy

Version: 1.0
Due Date: 30/04/2016
Delivery Date: 30/04/2016
Nature: Report
Dissemination Level: Public

Lead partner: Siemens
Authors: Leonard Pitu (Siemens), Paula Ta-Shma (IBM), Shelly Garion (IBM), Achilleas Marinakis (NTUA)

Internal reviewers: Juan Rico (ATOS), Orfeas Voutyras (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	30/03/2016	Leonard Pitu	Siemens	Internal draft starting from D3.1.2
0.2	5/04/2016	Leonard Pitu	Siemens	Review motivation & requirements
0.3-0.4	16/04/2015	Leonard Pitu	Siemens	Node-Red components; Updates to HW parts
0.5	20/04/2015	Leonard Pitu	Siemens	Introduction
0.6	21/04/2015	Achilleas Marinakis	NTUA	Fuzzy Privelets
0.7	22/04/20150	Shelly	IBM	Consent Management
0.8	25/04/2015	Leonard Pitu	Siemens	Conclusions
0.9	27/04/2015	Leonard Pitu Paula Ta-Shma Shelly Garion Achilleas Marinakis	Siemens IBM IBM NTUA	Pre-final version
0.95	26/04/2016	Juan Rico	ATOS	Internal review #1
0.96	27/04/2016	Orfeas Voutyras	NTUA	Internal review #2
0.97	29/04/2016	Leonard Pitu	Siemens	Minor changes
1.0	30/04/2015	Leonard Pitu Paula Ta-Shma Shelly Garion Achilleas Marinakis	Siemens IBM IBM NTUA	Final version

Table of Contents

Executive Summary 8
1. Introduction 9
2. Motivation and Overview 10
3. Requirements 15
   3.1. Risk Analysis 15
   3.2. Exploitable Features 21
   3.3. Requirements 23
   3.4. Trust 29
   3.5. Security 30
   3.6. Privacy 32
4. Security Assessment and Standardization 33
   4.1. Standards & guidelines 33
   4.2. Assessment and Certification 34
5. Architecture 36
6. Security Components 39
   6.1. Hardware-coded security 39
   6.2. Cloud storage security 53
   6.3. Cloud storage privacy and consent management 56
   6.4. Privacy 60
7. Conclusions 68
8. References 69

Table of Figures

Figure 1: Intrusion model 10
Figure 2: Vulnerability life cycle 11
Figure 3: Data Breach Investigation Report 13
Figure 4: Attacks types 13
Figure 5 Communication Security Model 31
Figure 6: COSMOS security architecture 37
Figure 7: ZC702 Board Block Diagram 40
Figure 8: FPGA Fabric Architecture 40
Figure 9: AES Module Internal Structure 43
Figure 10: AES Timing Diagram 44
Figure 11: AES Module Functionality Chart 45
Figure 12: Diffie-Hellman key agreement 47
Figure 13. ECC operation hierarchy 48
Figure 14: Top level schematic of the ECDH sub modules 49
Figure 15: ECDH module structure 50
Figure 16: Noise generator 51
Figure 17: Transistor noise 51
Figure 18: "noise controlled" LM555 51
Figure 19: Output Signals 52
Figure 20 Node-RED components 53
Figure 21: Guardium Monitoring Data Flow 55
Figure 22: Swift Auditing Architecture 55
Figure 23: Consent management architecture in COSMOS 58
Figure 24: Privelets Configuration File 63
Figure 25: Follower's list 63
Figure 26: Symmetric Trapezoidal Membership Function 64
Figure 27: VE JSON data 65
Figure 28: VE JSON message 66

List of Tables

Table 1: STRIDE analysis 15
Table 2: DREAD analysis 16
Table 3: Firmware Update 21
Table 4: Code Execution 21
Table 5: Storage 22
Table 6: Physical Intrusion 22
Table 7: Cloning 22
Table 8: Security Requirements 23
Table 9: Interface signal description 42

Table of Acronyms

Acronym	Meaning
API	Application Programming Interface
ARM	Architectural Reference Model
ADC	Analog-to-Digital Converter
D	Deliverable
DAM	Database Activity Monitoring
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
IPC	Inter-process communication
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
RPC	Remote Procedure Call
SHA	Secure Hash Algorithm
SPI	Serial Peripheral Interface
SoC	System on Chip
SSH	Secure Shell
VE	Virtual Entity
WP	Work-package
STRIDE	Spoofing of user identity; Tampering; Repudiation; Information disclosure (privacy breach or data leak); Denial of service (D.o.S); Elevation of privilege
WSGI	Web Server Gateway Interface

Executive Summary

This report (D3.1.3) presents the updated version of the documents (D3.1.1/D3.1.2) describing the overall security architecture of COSMOS. This document extends the previous released documents in the areas of risk identification and analysis, HW security components, cloud security and privacy. Major differences as compared to D3.1.2 are:

- Inclusion of a statistical unit cu analyze the random number provided by the external random number generator
- Development of a high level, graphical user interface to facilitate the usage of hardware security components
- Development of new privacy components allowing for fine-grained data obfuscation
- Development of a consent management framework.

COSMOS' main objective, as it is specified in the DoW, is the delivery of dedicated mechanisms which enable smarter and safer "things" in the IoT domain.

In synergy with WP2 – Requirements and Architecture – WP3 provides guidance for all other WPs as to how developers should implement their components in order to meet both performance and security criteria. Based on identified risks, use-cases and requirements set out in WP2, WP3 enhances COSMOS with dedicated security components and mechanisms which are designed to raise the overall system without hindering performance or 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.

The present deliverable together with D2.3.2 helps COSMOS meet the "secure by design" goal.

Security is known to have 5 pillars (confidentiality, integrity, availability, authenticity, non-repudiation) which are addressed by the COSMOS functional components. In this context, COSMOS strongly focuses on data integrity, authenticity and non-repudiation in the context of hardware-coded and cloud security.

The present report serves as the second milestone in developing the COSMOS security enhanced architecture. It 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 according to the identified use-cases and requirements. The present report reflects inputs collected from all project partners and synthesized into the generic COSMOS Security Architecture.

Furthermore, 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.

1. 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 encompass the COSMOS vision of end-to-end security and privacy which ranges from the smallest components up to the cloud platform. The COSMOS security concepts cover hardware and software components, data generation and consumption mechanisms and communication protocols. The final goal is to use security mechanisms to enable the vision of COSMOS in a secure and trustworthy environment 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 (i.e. 2015) presented by the recent leaks covering PRISM and other “Big Brother” like programs. Present day security attacks not only pose high risks but also provide insight into what the future holds: a world more security aware in which every flaw can be exploited given enough time and resources. Attackers are motivated by material gains 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 that preserve privacy. Such approaches should address lower levels of IoT environments (i.e. hardware-coded techniques) as well as data management, application levels and user management. Tamper-resistant smart devices, dynamic and evolutionary trust models, secure data stores, applications with built-in security and privacy are critical for sustainable smart city applications.

Personal privacy, trust and security are of crucial importance for both end-users and system providers. Trust is lost quickly 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.

2. Motivation and Overview

In this section we describe the main ideas which serve as the basis for our activities and are the fundamentals for implementing the project's vision. As such, security concepts and the respective attacks have to be defined.

A basic terminology for security and dependability was developed during the MAFTIA project where security breaches led 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.

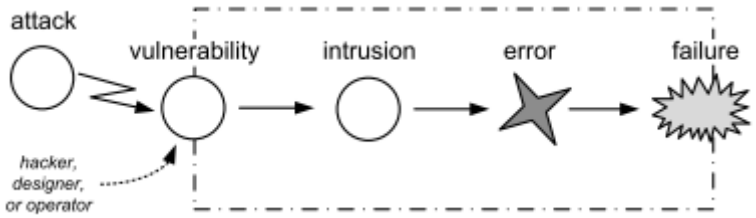


Figure 1: Intrusion model

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".

The terminology related to security attacks is defined as follows [1]:

- **Attack**: a malicious interaction fault, 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.

A successful attack attempt results in an intrusion as illustrated in Figure 1 [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.

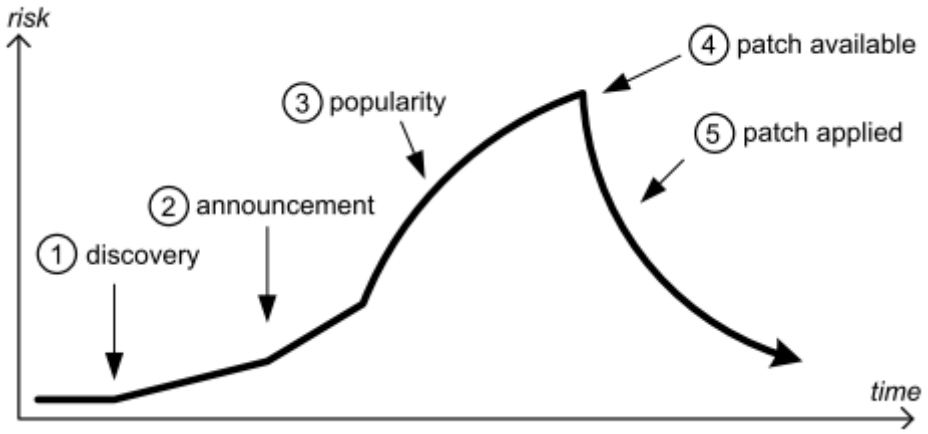


Figure 2: Vulnerability life cycle

Schneier developed a vulnerability life cycle in [3]. Figure 2 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 methods applied are 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 a users' identity to gain access to a system;
- **Denial-of-Service** – making the platform inaccessible to a user or group of users with the goal of causing service/financial losses;

- **Data integrity** – changing or altering data in order to reach a desired goal (e.g. obtaining profits or targeting a desired effect within the attacked platform);
- **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 [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 enabling 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 of time.

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).

The dream of IoT, a world of interconnected, smart devices, poses new opportunities and threats, both offering a great "playing ground" for developers and attackers. As the Data Breach Investigation Report [37] depicts, a growing number of security attacks can be observed in more than 95 countries around the world. The report depicts threats, culprits, targets, victims and the exposed attack.

As depicted in Figure 3, 92% of the identified attacks can be categorized and therefore quantified. 75% of the performed attacks are identified of being point-to-point and targeting a single aspect within a system or platform.

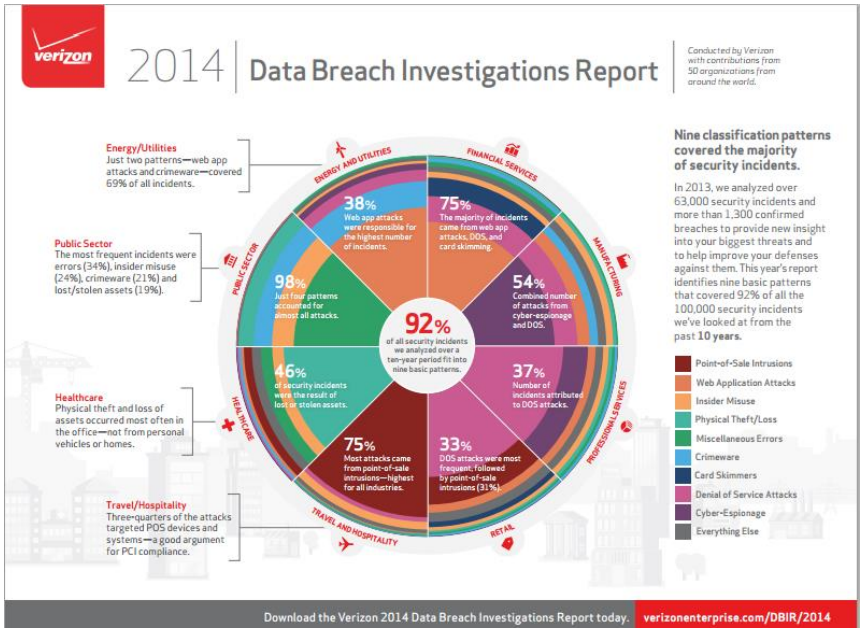


Figure 3: Data Breach Investigation Report

The number of security attacks is rising at an alarming rate as it can be seen in Figure 4. During the last years hacking has grown to being the number one security threat. This is due to the wide-spread of security "know-how" and therefore the identification of security breaches (over the last years security awareness has raised dramatically and proprietary solutions have become public knowledge leading to the discovery and, over time, strengthening of security levels) and on the other hand side due to public spread of attack tools which are providing playing grounds for new attacks.

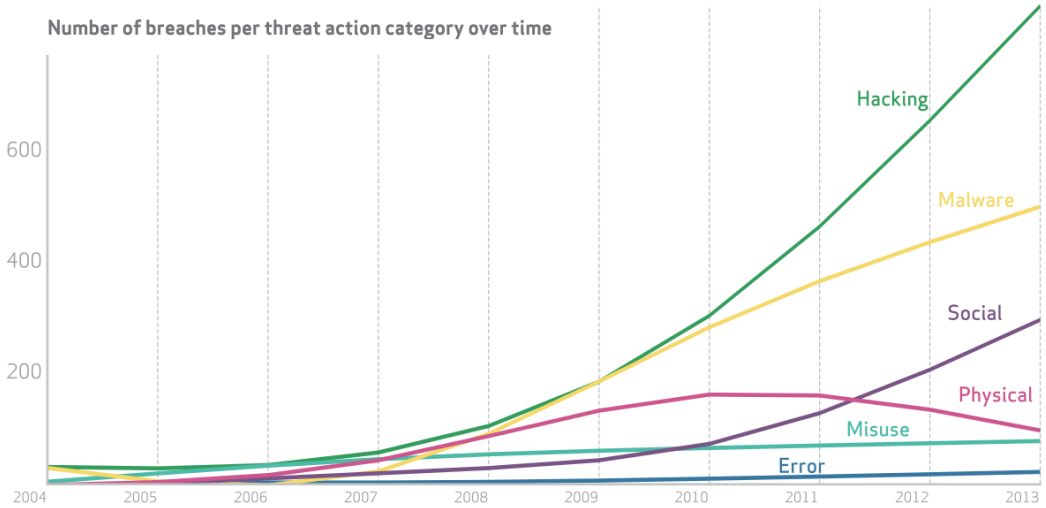


Figure 4: Attacks types

Taking the above considerations into account WP3 tries to provide answers to some of these security concerns. In this document we therefore present the risks identification and analysis according to the IoT-A, security requirements, important standards and assessment techniques. The document describes the COSMOS research activities in this scope which cover hardware security components, cloud storage security and privacy filters. The current document covers low level security (e.g. HW security or security of embedded "things") as well as high level security (e.g. privacy filters or cloud level security). 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. COSMOS facilitates IoT-based systems with end-to-end security and privacy, from hardware-coded approaches at device level, access control, encryption, multi-tenancy and cross-application mechanisms at the "data level", to the IoT services level with the injection of privacy –preserving mechanisms within things themselves.

3. 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. In order to realise these properties a risk analysis must be performed, as well as an identification of the most important attack points.

3.1. Risk Analysis

The risk analysis below is realised using the IoT reference architecture and depicts some of the most important traits of a platform. We have used commonly identifiable security threats derived from our use-cases as well as from technical literature and standards currently available.

Table 1: STRIDE analysis

	Human Actor	Privacy	Communication Channels	Devices
Spoofing Identity	Attacks may generate data on behalf of somebody else.	Identity theft. Human actor interacts with a malicious peer.	Access to restricted services.	Loss of device. Loss of correspondence between VE and physical device.
Tampering with Data	Attacks may lead to wrong/modified data being provided.	-	Wrong service calls. Wrong data push or pull requests.	Loss of control over a device (e.g. actuator). Loss of control over generated data and/or transmitted data.
Repudiation	No proof of integrity and originality can be made.	-	Local DDoS attacks cannot be traced back to their source.	Data is not correctly routed. Data is changed.
Information Disclosure	Theft of information and/or identity can occur.	Attacker gains knowledge of otherwise private information.	Access to restricted data (linked to spoofing identity).	Disclosure of device secrets. Possible changes in data path and/or data content.

	Human Actor	Privacy	Communication Channels	Devices
Denial of Service	Critical services may fail.	-	Service disruptions.	Device is disabled. DoS to an actuator.
Elevation of Privilege	-	-	Wrong authorization. Wrong information propagation.	-

Using the identified and assessed risks a simplified DREAD analysis is performed. DREAD stands for **D**amage, **R**eproducibility, **E**xploitability, **A**ffected users and **D**iscoverability. As in the IoT-A, the simplified analysis method uses 2 level (L – low, M – medium, H – high) to characterize each criteria. For each risk a mitigation plan is provided which is further discussed in D3.1.2 where the security and privacy architecture of COSMOS is described.

Table 2: DREAD analysis

Element to Protect	Risk	DREAD rating	Example of Causes	Mitigation Plan
Human Actor	Attacks may generate data on behalf of somebody else.	H/L/M/L/L	Spoofing attacks, theft of identity	Enforce strong security Enforce ACL Cryptographic protocols
	Attacks may lead to wrong/modified data being provided.	H/L/M/L/L		Enforce strong security Cryptographic primitives & protocols
	No proof of integrity and originality can be made.	L/L/M/L/L	Replay attacks	Enforce strong security Communication protocols Cryptographic primitives
	Theft of information and/or identity can occur.	D/L/H/L/L		Enforce strong security

Element to Protect	Risk	DREAD rating	Example of Causes	Mitigation Plan
	Critical services may fail.	H/M/M/L/L		Enforce strong security ACL Restricted access Self-healing capabilities
Privacy	Identity theft.	H/L/H/L/M	Credential theft Spoofing attacks Brute-force attacks Man-in-the-middle attacks	Enforce strong security Communication protocols Cryptographic primitives
	Human actor interacts with a malicious peer.	L/H/H/M/L	Redirection attacks	Enforce strong security Authentication Authorization ACL Message non-repudiation
	Attacker gains knowledge of otherwise private information.	M/M/M/L/H	Unprotected forms Wrong authentication policies	Enforce medium security Weak encryption Communication protocols Anonymity
Communication Channels	Access to restricted services.	H/L/M/L/L		ACL Security protocol
	Wrong service calls. Wrong data push or pull requests.	H/L/M/L/L		Enforce weak/medium security Data integrity checks

Element to Protect	Risk	DREAD rating	Example of Causes	Mitigation Plan
	Local DDoS attacks cannot be traced back to their source.	M/H/L/H/L		Enforce medium security Attack identification and localization schemes Attack source isolation
	Access to restricted data (linked to spoofing identity).	M/L/M/L/L		Enforce medium security Security policies Self-healing capabilities
	Service disruptions.	M/H/L/H/L		Enforce medium/high security MAC authentication Security schemes Communication protocols One time pad schemes
	Wrong authorization.	M/L/L/H/M		Enforce medium security Enforce cryptographic based protocols Enforce security policies
	Wrong information propagation.	M/L/L/H/M		Pattern identification Attack isolation

Element to Protect	Risk	DREAD rating	Example of Causes	Mitigation Plan
Devices	Loss of device.	L/L/H/L/L		Enforce weak security Enforce authentication schemes Use cryptographic credentials based on HW or SW cryptographic primitives
	Loss of correspondence between VE and physical device.	M/L/M/H/L		Enforce strong security Use ACL Use tamper detection mechanisms Communication protocols Authentication & Authorization schemes
	Loss of control over a device (e.g. actuator).	M/M/M/L/M		Enforce strong security ACL Strong cryptographic primitives (HW) Authentication & Authorization schemes
	Loss of control over generated data and/or transmitted data.	H/M/H/M/L		Enforce strong security Authentication & Authorization schemes Communications protocol

Element to Protect	Risk	DREAD rating	Example of Causes	Mitigation Plan
	Data is not correctly routed.	M/M/H/M/L		Enforce strong security Authentication & Authorization schemes Strong cryptography Communications protocol
	Data is changed.	H/M/H/M/L		Enforce strong security Authentication & Authorization schemes Strong cryptography Integrity checks
	Disclosure of device secrets.	L/L/L/L/H		Enforce medium security Use identity management
	Possible changes in data path and/or data content.	M/H/M/M/M		Enforce medium security Communication protocols Security policies

Each of these risks can be realised using many attack types as well as different efforts. Also these risks can pose threats to 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.

3.2. 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 devices 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 3: 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 unauthorized 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 4: 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
Buffer over/under-flow	Malicious code can be (remotely) executed	RAM hashing Secure Execution Encryption	System will detect and/or prevent buffer over/underflows
Code execution is altered by changing RAM memory content	At runtime variables are changed possibly changing execution flow; possible root access through shellcode	RAM hashing Secure Execution	System will detect RAM memory changes and react accordingly or will prevent changes at all

Communication packets are changed; DDoS attacks	Communication data is altered(e.g. metering data)	Encyption Authentication	Attacker is unable to decrypt the data and thus to alter its content

Table 5: 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 6: 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	HW security: • 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 7: 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 D2.3.2 [19].

3.3. Requirements

Given the above inputs WP3 has identified the following security requirements which the project tries to solve.

Table 8: Security Requirements

Requirement Type	Origin	Priority	Category	Description	Rationale	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	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	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	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	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	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.		
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.		

3.4. 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 trust 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. 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 – although almost impossible to achieve in practice we can build-up a trust pyramid or root-of-trust in which trust can be localized in a 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 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]. Although in this context physical intrusion (e.g. “hacking” a sensor in order to trigger a desired effect) are a threat COSMOS is not actively targeting these sort of attacks.
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, “impersonation”. For example, in the case of network 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 in the following sections.

3.5. Security

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

Although authentication and authorization are of crucial importance in any IoT/IT system, COSMOS is not actively targeting these primitives. There are a number of projects focusing on these aspects of security as well as well-established products and/or open source solutions. For these components COSMOS provides an architectural overview but on implementation level relies on existing solutions (e.g. OpenStack Keystone [38]).

3.5.1. 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 a communication security model is presented 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].

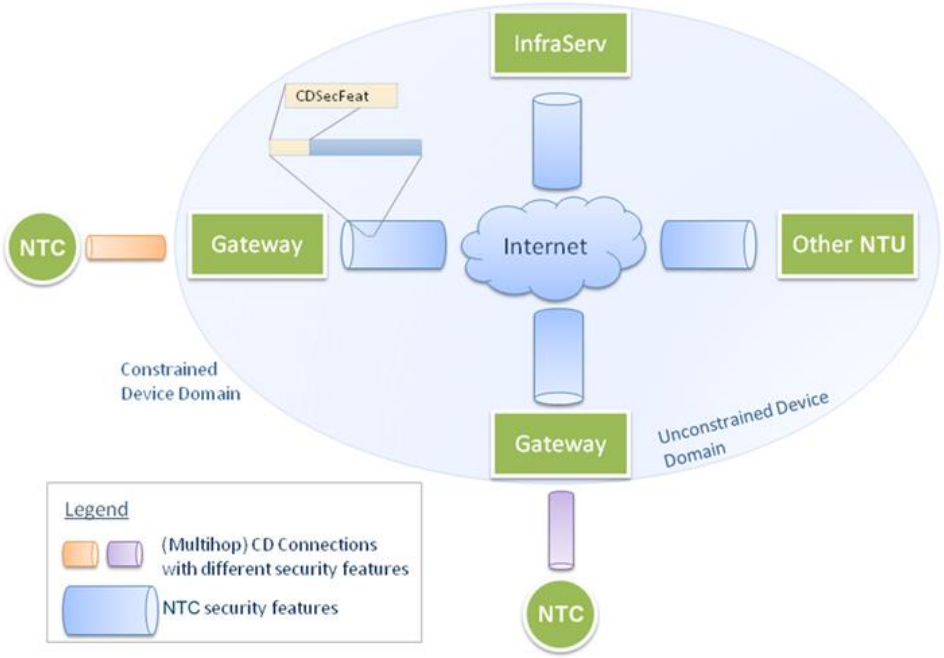


Figure 5 Communication Security Model

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

3.5.2. 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.

3.6. Privacy

The domain of privacy partially overlaps with 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 [46]. 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, use 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;
- 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. 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.

4.1. Standards & guidelines

4.1.1. 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.

4.1.2. 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.

The BSI publishes the most comprehensive assessment guideline of all having an extensive threat catalogue which provides extremely detailed assessments with respect to IT security. Even more, the BSI updates their catalogues on a regular basis therefore promoting IT security at a very high level.

4.1.3. 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.

4.2. Assessment and Certification

4.2.1. 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" are 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. Architecture

Taking the above factors into consideration the present section describes the COSMOS security architecture. As stated before, COSMOS' main efforts are oriented towards hardware security, cloud security and privacy at device level (i.e. privacy filters).

Components such as authentication and authorization are depicted in the current architecture but are not directly tackled by COSMOS. There are a number of research projects targeting authentication and authorization as there are commercial products and open source solutions. For this reason our security architecture is using ready-made components in order to implement authentication and authorization components. Therefore we have selected OpenStack's Keystone [38] in order to provide the authentication, authorization and key generation, distribution and management functions.

Given these facts COSMOS focuses on strong security at device and cloud level as well as privacy filters.

COSMOS is using a mixture of hardware and software components to realise the "end-to-end" security goal.

At VE level

Each VE in COSMOS needs to have a security marker in order to be used – there are 3 possible security marker types: unsecure (i.e. no security), secure (i.e. software only security) and highly secure (i.e. HW security). Data produced by the VE must also be privacy enriched that is the data must be "filtered" and private data removed according to the owner's rules.

At cloud level

Key management, generation and distribution as well as storage security are provided at cloud level. Cloud storage security is a key component to COSMOS' interaction model and is therefore data must comply with strict security regulations. The developed components also provide sandboxed execution environment at cloud level which strengthens the security even more.

At user level

All users suing COSMOS, either as developers or as end-users, must meet strict security regulations such as using unique keys and/or user name/password pairs.

The COSMOS security components 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 & Authorization: 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;
- Privacy: the information is property of the VE owner and therefore is privacy filtered.

Availability, non-repudiation and integrity are components covered by the hardware components on the device side and by the cloud storage components on the cloud side.

Privacy components are illustrated by the Privelets which enrich the VE's.

Authentication and authorization are covered by Keystone as well as key generation, distribution (on the cloud side) and management.

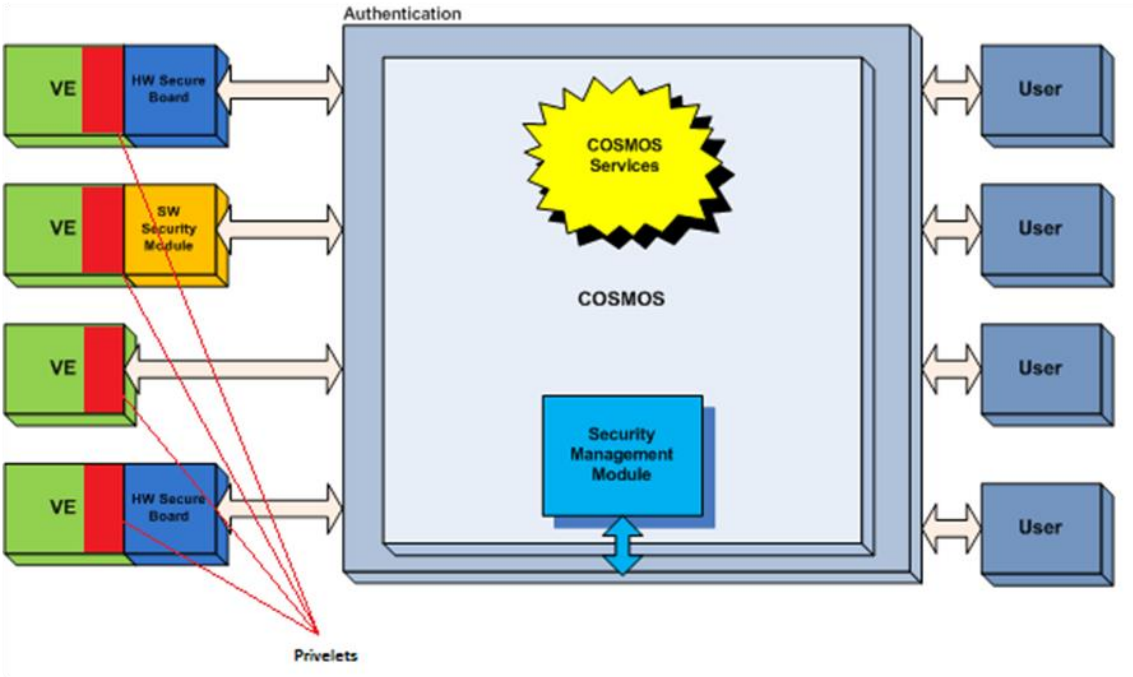


Figure 6: COSMOS security architecture

As depicted in Figure 6 the HW Security Board (or HW Board for short) is the practical implementation of a "thing" enriched with security and privacy components such as:

- HW encryption/decryption module;
- Key exchange module (i.e. Diffie-Hellman key exchange implementation);
- Hashing/checksum module;
- Privacy filters.

On the HW Board one or more VE's can reside, each with its own set of settings, security and privacy rules and communication protocols and/or interfaces.

On the cloud storage side the security is provided by Keystone and by the cloud storage components. Keystone is the project name for OpenStack Identity, a service that provides token, policy, and catalog functions via an OpenStack application programming interface (API). As is the case with other OpenStack projects, Keystone represents an abstraction layer. It doesn't actually implement any user-management functions; rather, it provides plug-in interfaces so that organizations can leverage their current authentication services or choose from a variety of identity management systems that are on the market.

Authentication is the process of establishing who a user is. Keystone confirms that any incoming functional call originates from the user who claims to be making the request. It performs this validation by testing a set of claims which take the form of credentials. The distinguishing feature of credential data is that it should only be accessible to the user who owns the data. It can consist of data only the user knows (user name and password or key), something the user physically possesses (a hardware token), or something the user "is" (biometric data like an iris scan or fingerprint).

After OpenStack Identity has confirmed the user's identity, it provides the user with a token that corroborates that identity and can be used for subsequent resource requests (for example to Swift object storage). Each token includes a scope that lists the resources to which it applies. The token is valid only for a finite duration and can be revoked if there is a need to remove a particular user's access.

6. Security Components

6.1. Hardware-coded security

6.1.1. Hardware Security Board

This section focuses on custom HW components which, at device level, are forming the so-called “root-of-trust”. These components are part of an embedded platform which implements an IoT device connected to COSMOS.

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, programmable logic, etc.)
- Low cost (for one prototype)

Having all these facts in mind, after the selection process it was chosen the Zedboard evaluation board. 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 Zedboard block diagram is shown below.

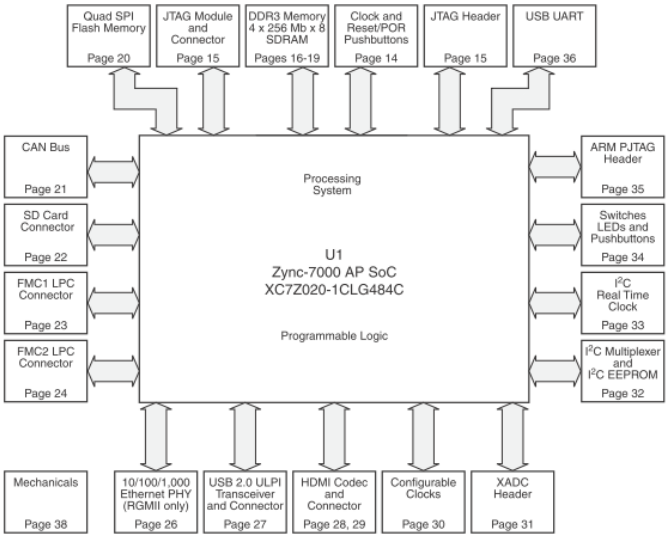


Figure 7: ZC702 Board Block Diagram

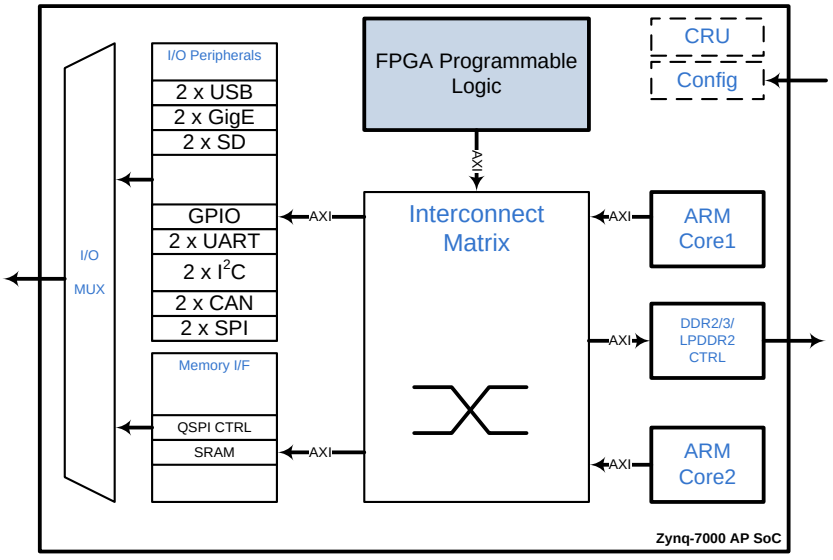


Figure 8: FPGA Fabric Architecture

The Zedboard 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 Zedboard 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 Zedboard 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 Zedboard 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 Zedboard 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.

6.1.2. Communication Interface

All peripherals within the FPGA fabric are attached to the CPU’s over the main system bus. In order to connect the APB to the AXI bus, a so-called bridge is needed. The used bridge is provided by the platform FPGA manufacturer (Xilinx) and is freely available in the tool-chain the demonstrator board was delivered with.

The table below describes the interface signals of the bus system. It is a standard APB slave interface.

Signal Name	Width	Dir	Description
APB Slave Interface			
clk_i	1	I	50 MHz clock

rstn_i	1	I	asynchronous reset, active LOW
psel_i	1	I	APB slave selection
penable_i	1	I	APB slave enable
paddr_i	8	I	APB address
pwrite_i	1	I	APB direction (WR=1, RD=0)
pwdata_i	32	I	APB write data
prdata_o	32	O	APB read data
pready_o	1	O	APB ready

Table 9: Interface signal description

Over the system bus there are 4 main interconnected modules:

- AES – implements the AES128 cryptographic algorithm
- ECDH – implement a Diffie-Hellman key agreement protocol
- TRNG – data acquisition module for the hardware true random number generator
- Statistical analysis unit – implement statistical and primacy tests for the generated random numbers.

6.1.3. AES

The implemented AES cryptographic accelerator is implemented using an APB bus system.

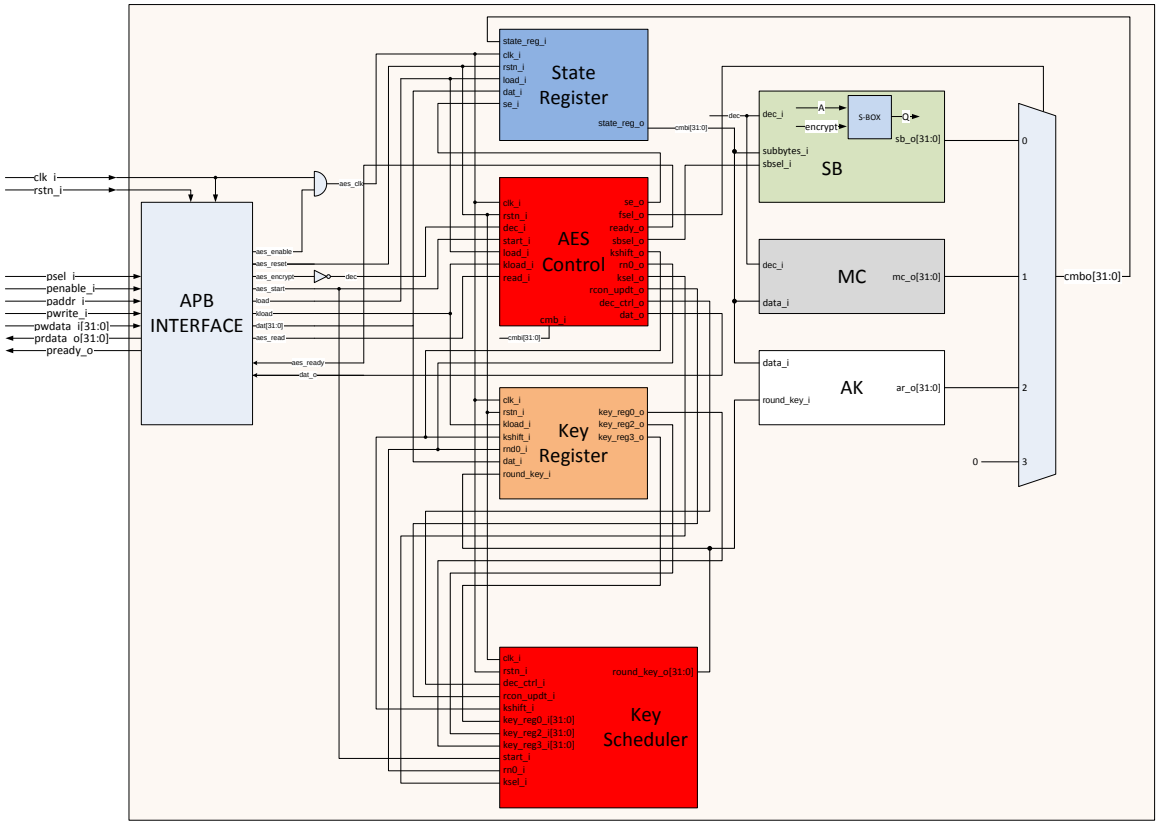
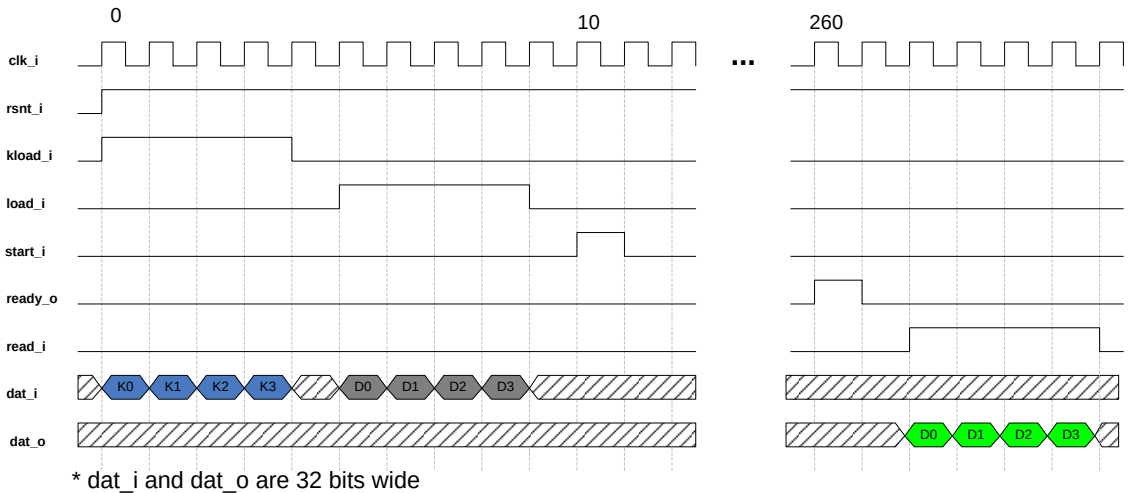


Figure 9: AES Module Internal Structure

The AES module is implemented according to the fips-197 standard and hard the internal structure as depicted in Figure 9. The module implements the basic building blocks the AES computing rounds rely on. The AES Control sub-module implements a state machine which controls the AES computation. The module implements both encryption and decryption according to the AES standard.



* dat_i and dat_o are 32 bits wide

** an encryption or decryption(without key expansion) takes 240clk

Figure 10: AES Timing Diagram

The timing diagram of the AES cryptographic accelerator is depicted in Figure 10. As the diagram shows the data and key are loaded over a 32bit bus and therefore need 4 clock cycles to load.

6.1.3.1 AES Functionality Description

Certain steps must be followed in order to encrypt/decrypt data correctly with the AES module described in this chapter (see Figure 11). After the chip Power On, the software has to enable the AES module – set bit [1] (*aes_enable*) of the **AES_CTRL** register. This bit gates the module's input clock. After the clock enable, a software reset must be provided by writing bit [0] (*aes_reset*) of the same control register. In this phase, the AES is initialized and enters the IDLE state, waiting for an encryption/decryption operation.

In case of an encryption, **AES_CTRL** bit [3] (*aes_encrypt*) must be set, and then 128 bits of data have to be first loaded into the **LOAD_DATA1…4** registers followed by writing 128 bits of key into the **LOAD_KEY1…4** registers. The encryption operation starts when bit [4] (*aes_start*) of **AES_CTRL** is set. This register bit is automatically cleared by hardware. Having the encryption started, the SW must poll bit [5] of **AES_CTRL** register (*aes_ready*) in order to detect the operation end. This bit is set by hardware and automatically cleared (by hardware) when 128 bits of result are read out from **READ_DATA1…4** registers.

In case of a decryption, **AES_CTRL** bit [3] (*aes_encrypt*) must be cleared, and then 128 bits of data have to be first loaded into the **LOAD_DATA1…4** registers followed by writing 128 bits of key into the **LOAD_KEY1…4** registers if the decryption keys are not the same with the encryption keys. The decryption operation starts when bit [4] (*aes_start*) of **AES_CTRL** is set. This register bit is automatically cleared by hardware. Having the decryption started, the SW must poll bit [5] of **AES_CTRL** register (*aes_ready*) in order to detect the operation end. This bit is set by hardware and automatically cleared (by hardware) when 128 bits of result are read out from **READ_DATA1…4** registers.

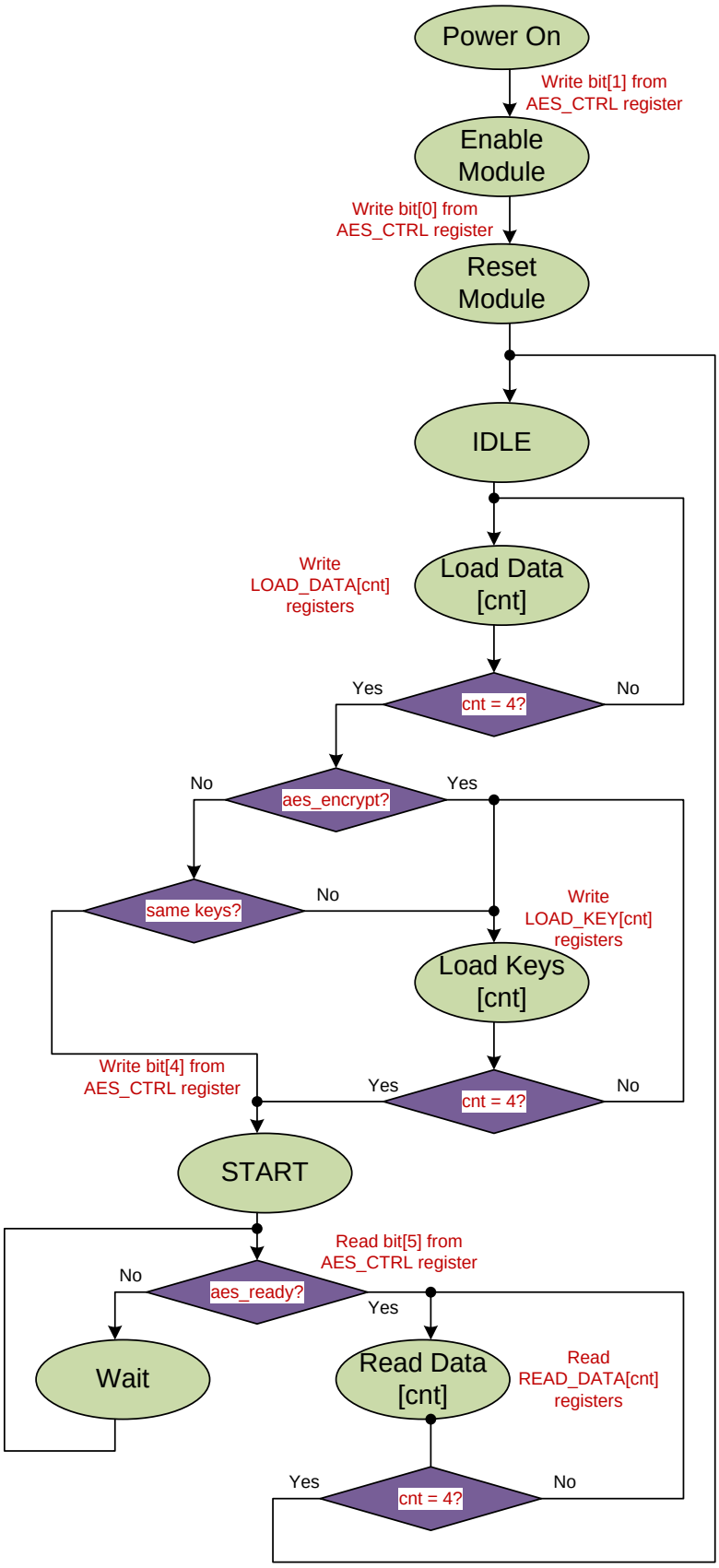


Figure 11: AES Module Functionality Chart

6.1.3.2 AES Address Space

The AES address space has been designed to allow further improvements. The data and control registers are directly addressable, by the CPU, using the implemented APB bus.

Start-Address	End-Address	Module/Memory-Name
0h	90h	AES

Module	Register/Memory	Read	Write	Address
/AES				
	LOAD_DATA1		w	0h
	LOAD_DATA2		w	4h
	LOAD_DATA3		w	8h
	LOAD_DATA4		w	Ch
	LOAD_KEY1		w	10h
	LOAD_KEY2		w	14h
	LOAD_KEY3		w	18h
	LOAD_KEY4		w	1Ch
	READ_DATA1		w	20h
	READ_DATA2		w	24h
	READ_DATA3		w	28h
	READ_DATA4		w	2Ch
	AES_CTRL	(r)(h)	(w)	90h

6.1.4. Diffie-Hellman Key Exchange

In order to use the AES hardware module, an encryption key is needed. This key is to be generated in the COSMOS platform and published to the HW Security Board. For this reason a hardware implementation of Elliptic Curve Diffie-Hellman is needed. Diffie-Hellman is an anonymous key agreement protocol that allows two parties, each having an elliptic curve public–private key pair, to establish a shared secret over an insecure channel [21], [22], [23], [24].

The present implementation is focused on developing a hardware prototype of a Diffie-Hellman key agreement protocol. The implementation considers only fixed-length 164 bits key and a standard APB interface [20].

6.1.4.1 Protocol Description

A general description of the Diffie-Hellman protocol between two entities Alice and Bob:

- Alice and Bob agree on a finite cycle group (in this case a Galois group) and a generating element, a point on the elliptic curve. These parameters are usually recommended in NIST or Certicom documents [28];
- Alice chooses a random natural number a and sends $a * P$ to Bob;
- Bob chooses a random natural number b and sends $b * P$ to Alice;
- Alice computes the secret shared key $a * (b * P)$;
- Bob computes the secret shared key $b * (a * P)$.

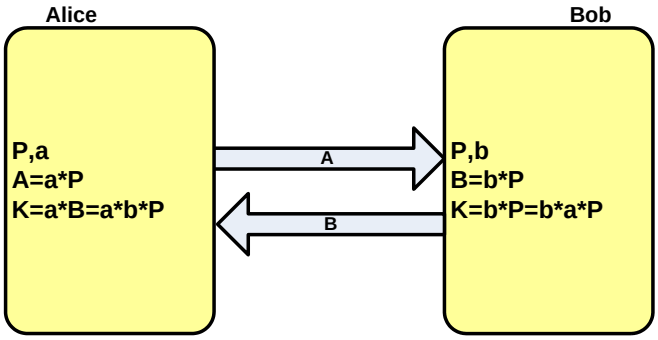


Figure 12: Diffie-Hellman key agreement

6.1.4.2 ECC / ECDH Operations

Elliptic curve cryptography is a public-key scheme based on elliptic curve over finite fields. In particular the present hardware implementation uses Galois Field F_{2^m} [26], [27], [28], [29], [30], [31], [32].

The simplified version of the elliptic curve E over F_{2^m} is:

$$y^2 + xy = x^3 + ax^2 + b \quad (1)$$

Where:

$$x, y, a, b \in F_{2^m}. \quad (2)$$

A private key n is randomly selected from $[1, 2^m]$. The public key Q is computed by $n * P$, where P and Q are points on the elliptic curve.

For the strength of the algorithm the parameters (a, b and P and the primitive polynomial for generating the Galois Field $t(x)$ must be correctly choose. The NIST documentation [32] recommends the following values:

$$a = 1 \quad (3)$$

$$P_x = 2\ \text{fe13c053 7bbc11ac aa07d793 de4e6d5e 5c94eee8} \quad (4)$$

$$P_y = 2\ \text{89070fb0 5d38ff58 321f2e80 0536d538 ccdaa3d9} \quad (5)$$

$$t(x) = x^{163} + x^6 + x^3 + 1 \quad (6)$$

Computing $n * P$ is denoted as scalar multiplication and it is based on point addition and point doubling operations over elliptic curve. Considering the scalar n in binary format the algorithm works as follows:

$$n = n_0 + 2 * n_1 + 2^2 * n_2 + \ldots + 2^m * n_m \; where \; [n_0 .. n_m] \in \{0,1\} \quad (7)$$

The formulas for point addition and point doubling over elliptic curve are presented below:

$$R = P + Q \quad (8)$$

$$x_R = \lambda^2 + \lambda + x_p + x_q + a \quad (9)$$

$$y_R = \lambda(x_p + x_R) + x_R + y_p \quad (10)$$

$$\lambda = \frac{y_Q + y_P}{x_Q + x_P} \quad (11)$$

$$R = 2P \quad (12)$$

$$x_R = \lambda^2 + \lambda + a \quad (13)$$

$$y_R = x_P^2 + \lambda x_R + x_R \quad (14)$$

$$\lambda = x_P + \frac{y_P}{x_P} \quad (15)$$

The pseudo-code below is a brief description of the implemented ECC algorithm.

$$Q \coloneqq 0$$
$$for\ i\ from\ 0\ to\ m\ do$$
$$if\ n_i = 1\ then$$
$$Q \coloneqq Q + P\ (using\ point\ addition)$$
$$P = 2P\ (using\ point\ doubling)$$
$$Return\ Q$$

These formulas are based on operations in Galois field: addition, multiplication and inversion. The ECC operation hierarchy is depicted in Figure 13.

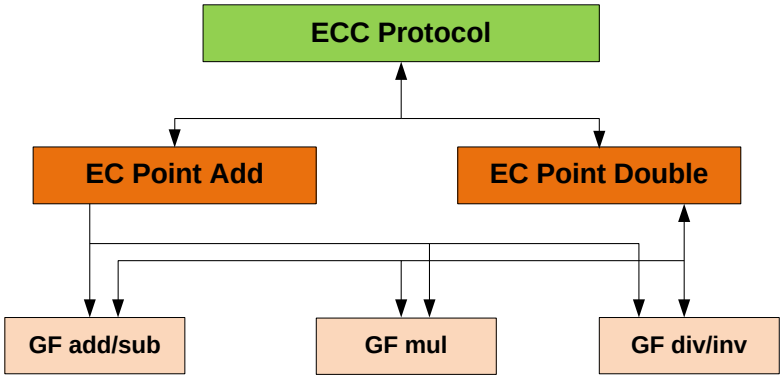


Figure 13. ECC operation hierarchy

In the following chapters the ECC hardware sub-modules are presented.

6.1.4.3 ECDH Interface

The top level of the ECHD hardware implementation is shown in Figure 14. The ECDH module has a custom interface which is adapted, via a Control Interface, to a standard APB bus system. The ECDH Control Interface also contains the Register Interface which is addressable via the APB.

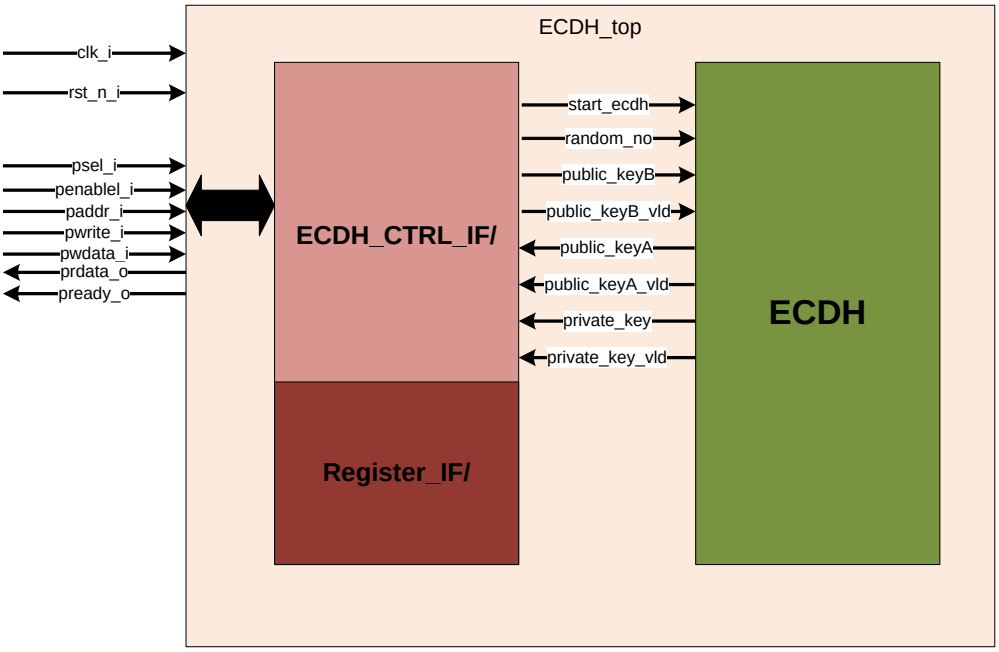


Figure 14: Top level schematic of the ECDH sub modules

6.1.4.4 ECDH Module

The ECDH module represents the core of the elliptic curve Diffie-Hellman key agreement protocol. It computes the secret share key between the two parties based on the chosen random number and the public key received from the counterpart.

ECDH consists of:

- Interface – custom interface & protocol for easy debug;
- Glue logic – synchronizations; flow control;
- ECC:
  - Point Addition;
  - Point Doubling;
  - Glue Logic (registers, synchronization, flow control).

The structure of ECDH module is shown in Figure 15.

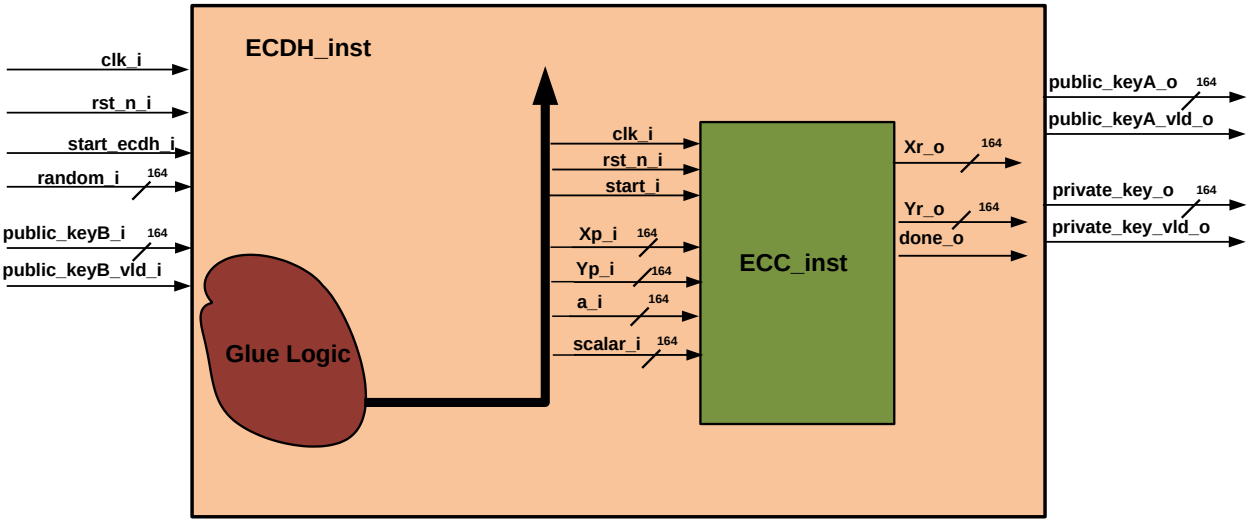


Figure 15: ECDH module structure

6.1.4.5 HW Random Generator

In order to provide the proper level of security, the ECDH module needs a random seed – a true random number generator. A TRNG can only be realised using a natural occurring phenomena which cannot be controlled, modelled or anticipated.

There are a great number of TRNG implementations but most are too complex, therefore too expensive to implement in low-cost, low-power hardware devices, such as those envisioned by the IoT.

For this reason, we started from the ground up, with the most basic component which lies at the foundation of modern electronics, the transistor. Due to the physical structure of the transistor, the fabrication process and the materials it is made out of, transistors are characterized by a natural occurring noise. For this reason we used the following components as depicted also in Figure 16:

- Noise device – a base-emitter junction of a discrete bipolar junction transistor biased at a small dc current (few micro-amps to tens of micro-amps);
- Amplifier – high frequency voltage amplifier (voltage gain of 4 to 60);
- VCCS (or VCIS) – voltage controlled current source (a current source for a grounded load, with a current of a few mA controlled by the input voltage – the amplified noise);
- OSC – an oscillator realized with a classic LM555 timer, based on a capacitor charged by the current source (and discharged by a hundred ohm resistor through the discharge transistor of the 555 resulting an oscillation frequency of a few hundred kHz);
- Output stage (or Output Buffer) – consists on a level shifter and a voltage follower.

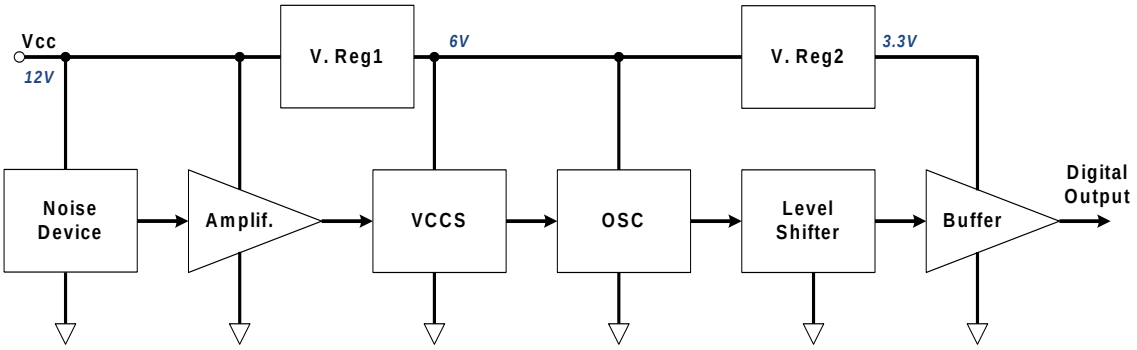


Figure 16: Noise generator

The reverse bias base-emitter junction of a bipolar junction transistor generates noise, with peak-to-peak amplitude of tens to hundreds of mV, (red wave in Figure 17) that is amplified by the high frequency amplifier (yellow line in Figure 17).

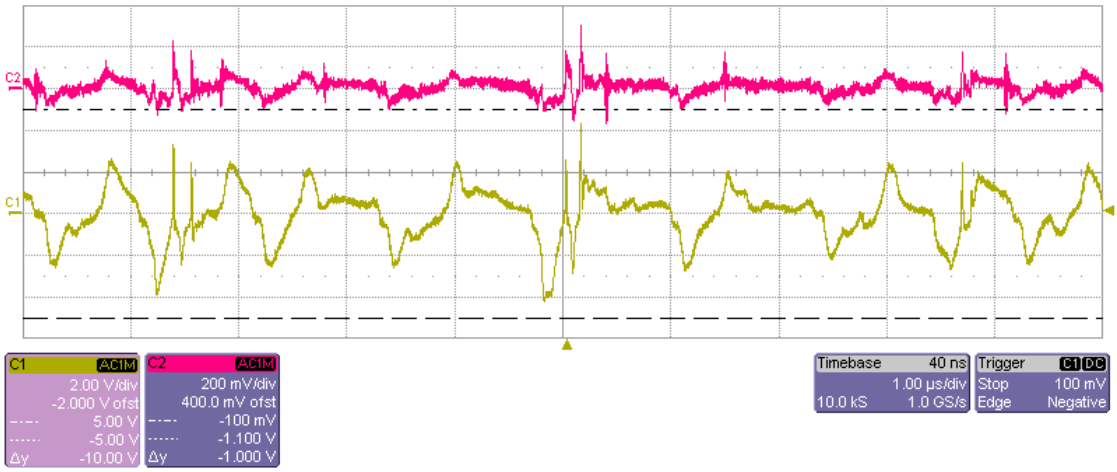


Figure 17: Transistor noise

The amplified noise is applied to a current generator that is used to charge the capacitor of a square wave oscillator (realized with a LM555 circuit) such that the instantaneous frequency of oscillation is influenced by the transistor noise, as depicted in Figure 18.

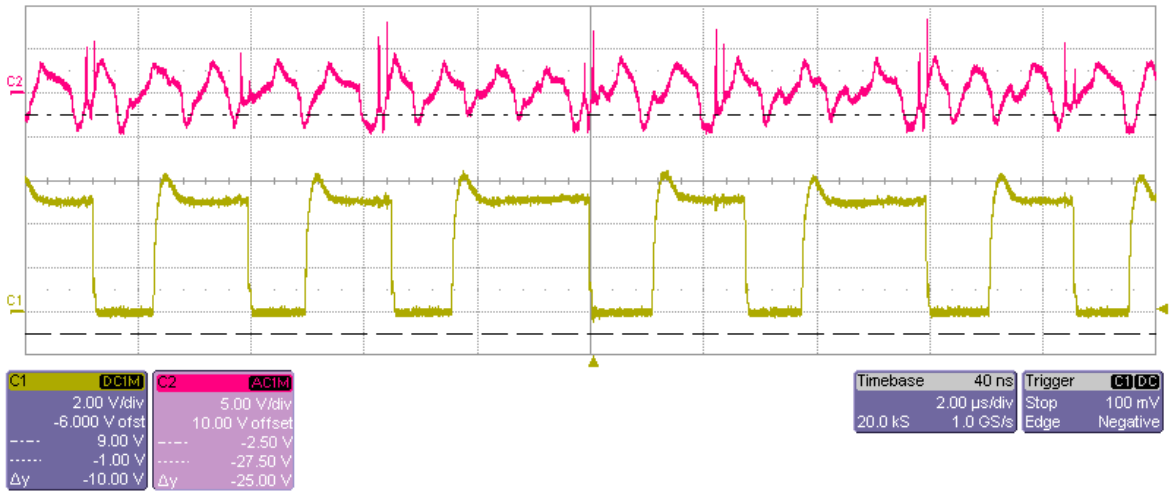


Figure 18: "noise controlled" LM555

The signal is applied to the output buffer that consists on a level shifter and a voltage follower as depicted in Figure 19.

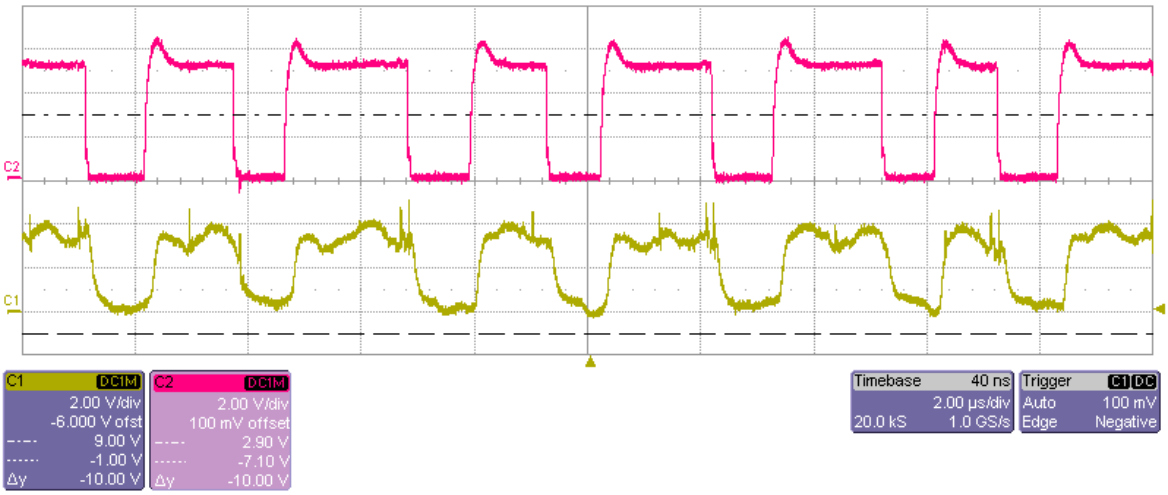


Figure 19: Output Signals

The output signal of the noise generator is further processed within the FPGA.

6.1.4.5.1 Prime number test unit

The developed random number generator is enhanced by a dedicated hardware testing unit which adds statistical testing functionality meant to identify attacks on the module as well as a prime number testing unit which is used to identify prime numbers.

Random number generators are usually subject to diverse security attacks. One of the most common attacks relies on injecting a known sequence of numbers which trigger the Diffie-Hellman key agreement protocol to agree a known key. During year 3 of the project we have focused on mitigating this type of attacks by means of a statistical analysis is performed over the generated numbers. The statistical analysis is able to identify repeating numbers or patterns within the random sequence. Samples which do not meet the necessary criteria, such as entropy threshold are removed and the random number generator is re-triggered.

Samples which pass the statistical tests are further passed through a prime number testing unit. Samples which are not prime are deleted and the random number generator is re-triggered.

For reference in developing both the true random number generator as well as the statistical tests the NIST provided test suites have been used [47].

The statistical unit has a modular design allowing for easy addition of further tests, which would extend the NIST provided suite as well as inclusion of further mechanisms to mitigate other attacks.

The module yields a 164bit long random prime number which can be further used in cryptographic computations such as encryption/decryption, key generation, key agreement and obfuscation.

6.1.5. Node-RED components

The above described components allow for much better performance while enhancing security, reliability and safety. In the IoT spirit the end-user and application developer is at the heart of any system therefore hardware components alone are not enabling this vision. For

this reason Node-RED nodes have been developed in order to facilitate the usage of the hardware accelerators.

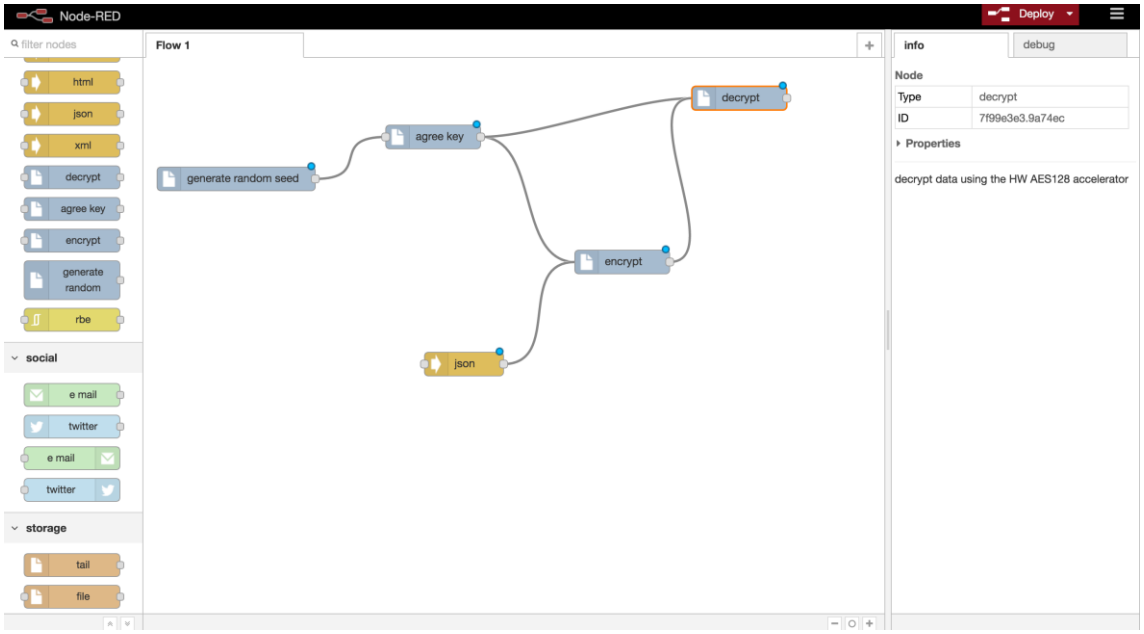


Figure 20 Node-RED components

The components available in Node-RED are:

- "generate random seed" – this component is used by both privacy and security mechanisms. The random number generator triggers the external TRNG as well as the prime number test unit implemented as a peripheral device within the programmable logic of the platform FPGA. This module yields a 164bit long prime number. There are no input parameters or configurations needed.
- "agree key" – is the node representing the Diffie-Hellman module. It uses the full hardware implementation of the Diffie-Hellman key agreement protocol. The node has two inputs – one for the random number and one for the key agreement messages as depicted in section 6.1.4.1.
- "encrypt" and "decrypt" – these two nodes are used to encrypt and decrypt data. Each one of the nodes has 2 inputs, one for the key and one for data. Both nodes make use of the hardware accelerator which implements the AES cryptographic algorithm. Data block and key sizes must be exactly 128bit.

Each node calls the Python wrapper of the hardware components using the python-shell [48] library. The hardware components have already been memory mapped in order to facilitate high-level access.

6.2. Cloud storage security

Regarding cloud storage security, in year 2 we focused on activity monitoring and privacy and security policy management, since these are important areas for IoT. This has been handled by an area known as Database Activity Monitoring (DAM) and we extended this to object storage and researched how this technology can be used for IoT workloads. In year 3 we focused on consent management and this is covered in the next subsection.

Cloud-based storage includes inherent vulnerabilities as the cloud is a multi-tenant environment where resources are shared. Even within a single storage account there is potential data leakage between different enterprises units (e.g. human resources accesses

employees' code base, etc.). In practice, sharing storage hardware, such as in the case of OpenStack Swift object servers, is risky. Whether accidental, or due to a malicious hacker attack, data leakage would be a major security violation.

Since data protection is a core goal of information security, data store operators are required by regulations to keep a record of data accesses - logging file creation, reading, updating and deleting ("CRUD") activities for each user (i.e. an audit-trail). Uses of system resources and unsuccessful access attempts may also be logged. In other words, an audit trail keeps track of who did what, to what, and when they did it, as well as who tried to do something but was unsuccessful. Audit trails are a fundamental part of computer security, particularly useful for tracing unauthorized users and uses. They can also be used to assist with information recovery in the event of a system failure. It should be noted that an audit trail should allow identifying the sequence of actions made by a user potentially interacting with several services in distinct hosts.

As an example in the context of IoT, an audit-trail describing the temperature of a drug being transported from source to destination can be logged and audited per specific regulation (e.g. a regulation for allowed temperature for drugs being transported).

In COSMOS we extend DAM technology to also handle cloud storage. Specifically, we provide a full audit-trail kept in a secured external server, for data that is stored in Swift Object Storage arriving from the COSMOS VEs (via the message bus and the Dat Mapper). The approach allows us to monitor the data continuously and in real-time. In addition, we provide advanced DAM functions for Swift that allow proactively installing a policy in the Swift proxy layer so that it can enforce rules in real time as data is stored (e.g. Swift will be aware of the installed policy and regulations and will act as an active firewall for selected containers).

The design described here was implemented in year 2 and further implementation details can be found in deliverable D3.2.2.

6.2.1. Activity Monitoring for Swift

As defined in subsection 4.4.2.1 of Deliverable D4.1.2, Openstack Swift has two layers: A proxy layer in the front end and a storage layer at the back end. Users interact with the proxy servers that route requests to the backend storage layer. Swift is implemented using WSGI technology that allows plugging functionality into the request processor. Each request hitting a WSGI based server goes through a pipeline of such plug-ins, called middleware. For example, amongst the plug-ins that consist the pipeline at the proxy server are authorization middleware, quota related middleware and 'router' middleware that forwards the request to the appropriate server according to the location of the request target resource. In year 2, we demonstrated our approach for activity monitoring for Swift by integrating Swift with an IBM product called IBM InfoSphere Guardium, although the approach is generic and could be applied to other activity monitoring tools. In the longer term we would like to use our work to further research how activity monitoring tools and their integration with object storage can be successfully applied to IoT workloads and provide effective security and privacy monitoring and management for COSMOS.

IBM InfoSphere Guardium is a solution that addresses the entire database security and compliance life cycle with a unified web console, back-end data store and workflow automation system. Guardium enables to find and classify sensitive data in corporate databases (for which the database owner provided access for mining), assess database vulnerabilities and configuration flaws, capture and examine all database transactions, including local access by privileged users, monitor and enforce policies for specific data access patterns (such as sensitive data, privileged user actions, etc.) and to centralize the compliance

auditing process for enterprise compliance reporting, performance optimization, investigations and forensics.

Guardium auditing is provided by software S-TAP agents that are installed on the database servers and send a copy of the observed traffic to a Guardium collector (also called a Guardium Appliance). The following figure demonstrates the data flow in a typical Guardium installation that supports database activity monitoring (DAM). Note that the Guardium appliance may install a policy that is built via Guardium web interface to the S-TAP. By doing that, the S-TAP can act as a firewall (an authorization mechanism) that can block specific database requests as defined in the installed policy. The local S-TAP can be configured to operate in a disconnected mode in case of intermittent network disconnections to the appliance.

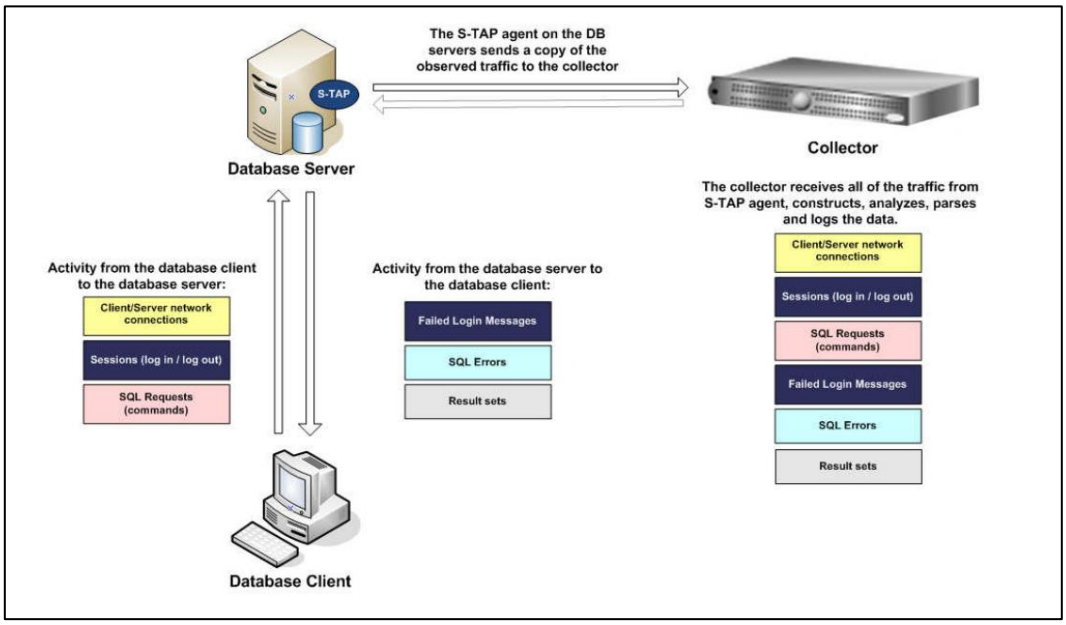


Figure 21: Guardium Monitoring Data Flow

In the context of COSMOS, in year 2 we created a Swift-Audit middleware and a Guardium-Swift agent (Swift S-TAP) that allow Guardium to maintain a full audit-trail of Swift activity and to enforce policies designed and deployed via the Guardium web interface. The following figure describes the chosen architecture.

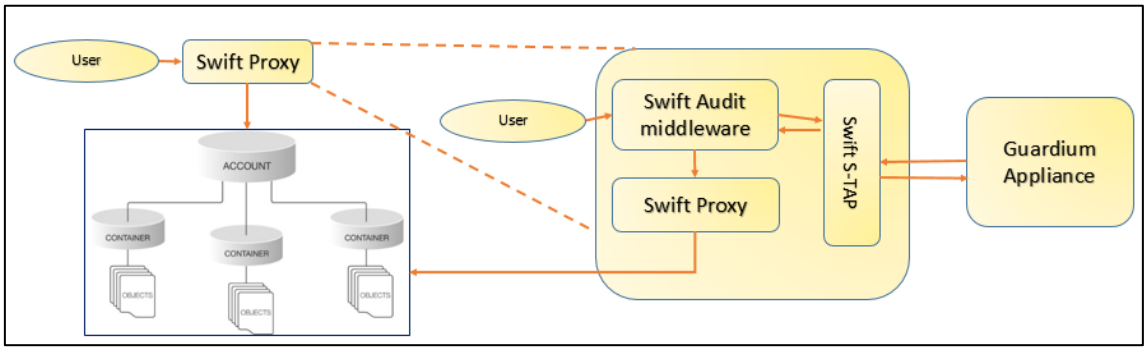


Figure 22: Swift Auditing Architecture

The left side of the figure presents a user interfacing Swift proxy to issue an I/O request (via REST API). The proxy interacts with the Swift backend, comprised of the account, container and object servers, to store the data on one of the disks. The right side of the figure describes

the internals of the Swift proxy extended with the Swift-Audit middleware and the Swift S-TAP that interfaces with the Guardium appliance. The architecture is designed so that the Swift-Audit middleware is generic and doesn't depend on a specific product such as Guardium for maintaining the audit-trail or to define the policies. Swift S-TAP will be a Guardium specific agent that communicates with the Guardium appliance via a Guardium defined protocol (called "Universal Feed v.1"). The protocol is based on Google's protobuf wire format. The following section defines the functional operation of each component.

6.2.2. Swift Audit Middleware

The Swift audit middleware is implemented using WSGI technology. It is a python component that receives every request arriving at the proxy. The middleware is responsible to collect the required information related to the request. For example, the middleware should extract the user that is issuing the request, the timestamp, the target container, the target object, the authentication token used for the I/O request, etc. Once the related information is collected that middleware passed the information to the Swift S-TAP agent for further processing. The middleware will use an IPC mechanism to invoke an RPC like method that will determine whether to block the request, or pass it on (e.g. an authorization mechanism similar to the database S-TAP mentioned in subsection 6.2.1). Note that the middleware is generic and is not aware of a specific audit-trail service. It is merely a bump on the wire, it extracts the relevant information and invokes a pre-configured agent (defined in the swift proxy configuration file) waiting for the outcome of the request.

6.2.3. Swift S-TAP

The Swift S-TAP agent is the proxy to a specific audit-trail implementation. On one hand it receives an event coming from the Swift audit middleware, on the other hand it interacts with a specific audit-trail service that receives events (e.g. audit events) and provides policy rules to be enforced on specific events. Specifically, each event received by this agent is checked against the set of rules previously installed. According to the rules, an event may be sent to the appliance for auditing, or the request may be blocked or ignored. Note that not all events will require an audit event, typically the security officer that defined the rules will determine which storage related events will be of any interest and will require logging or special auditing.

6.3. Cloud storage privacy and consent management

The Internet of Things (IoT), Cloud, Mobile and other technologies are resulting in the collection of more and more data, and increasingly sensitive data. While this data can be a gold mine for enterprises, it can also constitute a major risk for them. Legislation and privacy norms are becoming increasingly strict with regard to collection and processing of personal data, requiring informed consent of individuals for processing their data for specific purposes. According to these legislation and norms (see for example [42]), data may be processed and analysed only when the data subject has given his/her consent, or when the analysis is necessary for a contract, needed for compliance with a legal obligation, or is necessary for a task carried out for public interest, and the given consent must contain a (legitimate) purpose. Any data access attempt for which there is no consent for the specified purpose should be blocked. The purpose token can be used to validate the authenticity of the request.

IBM's consent management tool [40][41] provides a solution for automatic enforcement and auditing of privacy policies at multiple levels, including legacy applications and systems. With such a solution in place, both users and enterprises gain more control and confidence in what is actually being done with personal data within systems, and trust becomes a differentiator,

while auditing and compliance overhead is decreased for both the data processer and controller.

In COSMOS, smart heating application data is collected from various IoT devices (such as window activity and humidity sensors, boiler/mixed fuel output sensor, heating valve actuators) in order to help residents manage their heating schedule efficiently and control their consumption, taking under consideration various factors like their budget, comfort level and health. A flat can learn from its own experience or from the experience of the other flats, using Case Based Reasoning (CBR) techniques as described in D5.1.3.. However, due to legal restrictions, the residents consent should be given before processing and analyzing their data. Our aim in year 3 is to integrate IBM's consent management tool in the COSMOS smart heating application.

6.3.1. Consent management

IBM's existing consent management tool is composed of:

1. **A Consent Manager service on Bluemix** that manages everything related to the consent governing the use of data in the enterprise. It is responsible for the collection, storage, and maintenance of privacy policies and user consent. It also handles the logic of deciding whether a data item can be accessed in a certain context as is, whether it must be obfuscated in some way before being released, or whether it cannot be released at all.

We use the concept of "consent templates" to define the parameters of consent for a specific service provided by the organization, based on which specific consent is collected from the service users.

An initial version of the Consent Manager service includes:

- Support for purpose based user consent
- Consent per data item corresponding to the purpose and service
- Purpose-based access control
- REST APIs for registering organizations and users, creating consent templates for services, and creating specific consent contracts
- Basic UI for consent template definition
- Basic logging & reporting solution

2. **Data Access Controller** which is responsible for controlling access to the data and enforcing the data privacy policies, based on the decisions made by the Consent Manager. This component is highly dependent on the specific data store, data format and application accessing the data.

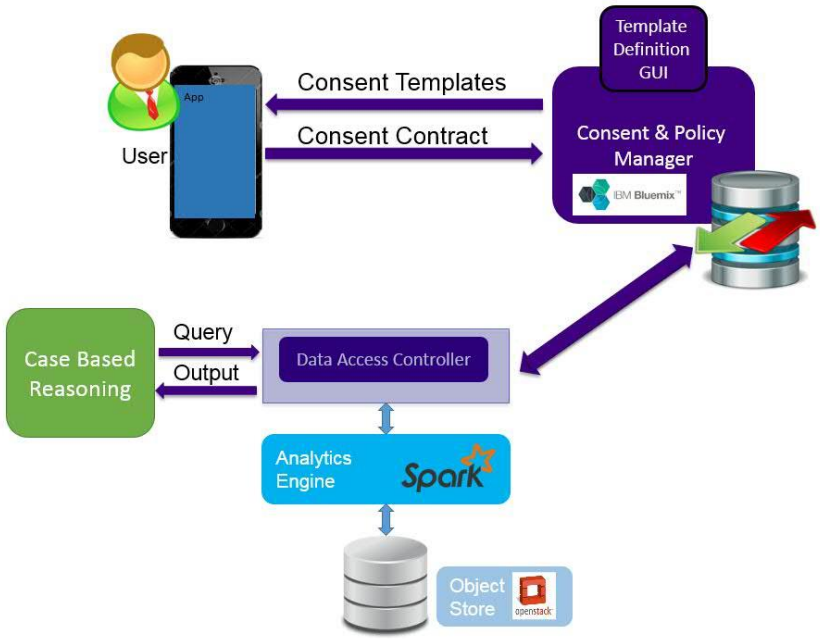


Figure 23: Consent management architecture in COSMOS

6.3.2. Consent management architecture in COSMOS

The consent management architecture in COSMOS, described in Figure 23, will contain the following components:

1. **IBM's Consent Manager** service on Bluemix (already available, see [43], [44]).
2. **Home Owner Consent GUI:**

   Presents the consent template and purpose to the end-user and obtains the consent contract, e.g. a consent screen that will be provided on the tablet to the end-users by the housing security officer.
3. **Data Access Controller:**

   We plan to implement a reference Data Access Controller for SQL-based data access, as a separate layer within the accessing application, based on Apache Spark and OpenStack Swift object store. It will contain the following components:

   a) Consent Socket Client: obtains the SQL query and the token for the purpose from the CBR application (using a UDP socket or HTTP/HTTPS RESTful API), and outputs the result after processing it (preferably in JSON format).

   b) A filtering logic, including parsing the SQL request and response, and replacing values of items in the response that are not consented. If a user has completely opted out of a service, his/her record is removed from the result set. Otherwise, specific fields are "nullified" according to the field type (e.g., empty string, 0 for integers, null for objects, etc.). Other anonymization techniques will not be supported in the demo.

   c) Communication with the consent manager service, that compares the token for the purpose with the user's consent, and obtains the decision about if and how the data may be used for the stated purpose.

   d) Consent Storage Connector: sends the SQL query to Spark SQL, and obtains the corresponding data records stored in OpenStack Swift.

Due to the filtering logic of the SQL queries, there are several assumptions on the structure of the SQL queries:

1. Fields are selected using the following format:
   SELECT <table_name>.<field_name> AS <field_name>
   (for example, SELECT Activity.activityId AS activityId)
2. SQL keywords are in capital letters (SELECT as opposed to select).
3. It only takes into account the fields that appear before the FROM keyword (and not fields that are part of the WHERE or GROUP BY clauses). Such queries are allowed, but fields that appear after FROM will not be checked for the user's consent.
4. It assumes that the result records contain a unique identifier of the user (e.g. hid) that is also known to the consent manager.

6.3.3. COSMOS smart-heating use-case scenario

Data is collected from various IoT devices in the COSMOS smart-heating use-case scenario, such as window activity and humidity sensors, boiler/mixed fuel output sensor, heating valve actuations, and its structure is given in the following schema:

- Personally identifiable information (PII), such as: name, address of the meter/sensor, phone, e-mail address, postal address. Due to its sensitivity, such data is not available in COSMOS testbed, and we plan to synthesize it for the demo.
- Time series data of the smart meter and sensor reading:
  - estate
  - hid – a unique identifier of the flat, that will also be known to the Consent Manager service.
  - timestamp (ts)
  - heatmeter reading:
    - instant, flowRate, flowTemp, returnTemp, ts, cumulative
  - sensor reading:
    - type, state, ts, sid

The CBR application (see [45] for details) can access the data for the following scenarios:

i) Detect comfort levels in terms of flat temperature for a given resident (uses temperature sensor, window sensors, flowTemp/returnTemp of water flow, timestamp).

ii) Create automatic heating schedule (uses target temperature conclusion extracted from (i) or hardcoded and temperature sensor, flowTemp/returnTemp of water flow, and timestamp from data, to create the elementary cases from which the schedule will be built). Cases are extracted from access to the historical data.

iii) Damp prediction and identification (uses timestamp, humidity sensor, possibly window sensor, temperature sensor).

For (ii) the CBR application will access the historical data not only of the specific resident, but also of other residents that have given their consent to share their data, but only after a proper anonymization, namely the PII fields are removed, and the fields shared are only the ones mentioned in (ii) above, which are needed for producing the cases.

Similarly, consent of the resident for experience sharing between different users triggers the enablement of the experience sharing of the CBR, which is sharing between flats of the produced cases from (ii). Experience is created by the case extraction from historical data (stored in Swift) and then it is stored locally in the Virtual Entity (VE), where the resulting experience will be completely anonymized, namely the PII fields are removed, and the fields shared are only the ones that are included in the case format. For example, in the heating schedule app these would be initial inside Temp, final inside Temp after an interval and boiler actuation value (true or false), which is the processed information from the historical data.

If a user refuses for experience sharing between different users, but agrees to use the CBR application for his own local use, then Planner access to the historical data is enabled and the extracted cases will be stored locally in the Virtual Entity (VE). The exact way that the VE will obtain the user's consent for experience sharing (with other users or not) will be decided upon later. One option is to obtain it directly from the Consent Manager and another is to obtain it together with the historical data returned from swift.

For (iii) there is no experience sharing (during runtime), but it could have access to the historical data from multiple users.

The end-user will be able to choose between the following possibilities, when filling the consent contract:

1. **Default option:** do not expose data to CBR application at all
2. Expose data only for own, local use by the CBR application of the specific user (implies no experience sharing and no sharing of historical data to others). Namely, a user will learn only from his own historical data but not share it with others and not use historical data or experience sharing of others
3. Enable experience sharing between different users
4. Enable historical data sharing between different users

6.4. Privacy

6.4.1. Introduction

In the course of the COSMOS project, Task 3.3 is responsible for providing mechanisms enhancing privacy and trust of VEs, aiming at preventing personal information inference that the involved entities would wish to keep private. To this end, we developed a framework under which VEs use virtual identities to share information directly with their friends, with respect to the decentralized approach which is adopted for VE2VE communication within COSMOS.

In addition, we developed Privelets functional component, which acts as filter enabling VE developers to select which data are private and therefore non-shareable to other VEs or COSMOS platform and which are public. During Year 3 of the project, we enhanced Privelets component by adding the fuzziness functionality, giving the developers the capability to also tag a data field as fuzzy. In this case, the VE does not provide the real value of the field, but a fuzzy one, which is randomly selected within a range defined by the Application Developer and according to the privacy level (low, medium or high) of the specific field, defined by the VE Developer.

6.4.2. Privacy Mechanism

In this section we describe the whole framework under which any kind of VE2VE communication (including accessing a VE property, accessing an IoT-Service, Experience Sharing etc.) takes place.

The general idea, regarding this framework, is that VEs communicate with each other using a Peer-to-Peer Virtual Private Network, where each node (VE) has its own virtual IP address and therefore the real IP addresses (identities) remain private. This P2P VPN will be implemented based on FreeLan [33], which is a free, open-source and multi-platform tool. COSMOS platform acts as certificate authority in order to ensure the establishment of trust relationships between the interacting VEs.

Registration phase

When a new VE registers in the COSMOS platform, the latter is responsible for providing it with a virtual IP address as well as with the key and the signed certificates which are necessary to get a license to enter the VPN. The VE also obtains a unique ID ("PrivacyID"), which is exclusively used to communicate with the COSMOS platform privacy mechanism. The whole process is carried out by a server running at the COSMOS platform which:

- Receives the GET REST request from the VE that is interested in registering in COSMOS
- Assigns randomly a (unique) Virtual IP Address to the VE within the COSMOS VPN range (e.g. 10.0.0.1 – 10.255.255.254, if the VPN address is 10.0.0.0/8)
- Assigns randomly a (unique) PrivacyID to the VE
- Stores the mapping between the above information and the Real IP Address of the VE in a database
- Returns the information in JSON format

Configuration phase

Before entering the VPN, a configuration file (stored locally at the VE side) must be filled in, whose mandatory fields are the following:

- Real IP address (e.g. localhost:3030)
- COSMOS platform endpoint
- Virtual IP address (assigned by COSMOS platform during the registration phase e.g. 10.0.0.1/8)
- VPN address (e.g. 10.0.0.0/8)
- Path to private key
- Path to VE's signed certificate
- Path to COSMOS platform's certificate

VE Recommendation phase

We assume that VE1 wants to communicate with VE2, as a result of the centralized or decentralized VEs (Followees) recommendation functionality. In this case, COSMOS platform sends to VE1:

- The Virtual IP of VE2
- The request Key
- The response Key

This information is stored in the VE1 Followees' list (ontology).

Similarly, COSMOS platform sends to VE2:

- The Virtual IP of VE1
- The request Key (exactly the same as above)
- The response Key (exactly the same as above)

This information is stored in the VE2 Followers' list (ontology).

Similarly to the **Registration phase**, the keys' assignment is also implemented by a server running at the COSMOS platform and the mapping is stored in a database.

The keys are used in the authentication process which is described in section 6.4.4.2.

6.4.3. Privelets

Privelets is a functional component that runs at the VE side and therefore it has to be aligned with the others similar COSMOS components (e.g. Planner, Experience Sharing) in terms of resources, libraries etc. Consequently it uses:

- Java Runtime Environment version 1.8
- Mqtt Eclipse Paho project [39] Java client in order to:
  - publish data to Message Bus (push approach)
- Jetty server in order to:
  - handle POST or GET REST requests from other VEs (pull approach)
- Apache-Jena API [34] in order to:
  - read and write in the Followees' and Followers' list, which are both structured in OWL format

Configuration

The VE Developer is responsible for filling in the Privelets configuration file, structured in JSON format. An example of this file is shown below:

```
{
   "interval_between_requests":"10",
   "data":{
      "temperature":{
         "privacy_level":"low",
         "range":"0.1"
      },
      "id":{
         "privacy_level":"public",
         "range":"none"
      },
      "latitude":{
         "privacy_level":"medium",
         "range":"0.01"
      },
      "longitude":{
         "privacy_level":"high",
         "range":"0.001"
      },
      "domain":{
         "privacy_level":"public",
         "range":"none"
      },
      "ipAddress":{
         "privacy_level":"private",
         "range":"none"
      }
   }
}
```

Figure 24: Privelets Configuration File

The "interval_between_requests" key defines the smallest interval of time between two requests from the same VE that can be accepted. For instance, if VE1 sends a second request to VE2 within this time interval then the VE2 refuses to answer. The key applies to VE2VE communication and aims to protect the VEs, when acting as information providers, from repetitive requests that could probably overload their system. The unit of measurement is the second and the value is an Integer. The time of the last request (used in order to check for possible repetitive request) is stored as a DataProperty (in epoch format) in the Followers' list:

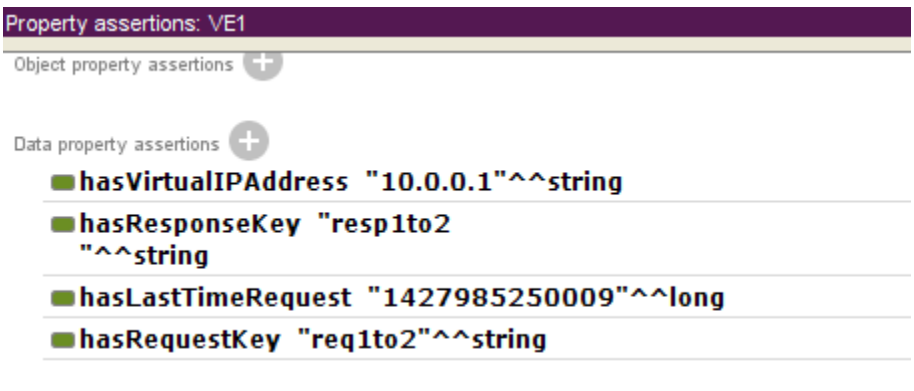


Figure 25: Follower's list

The “data” key refers to all the data that are generated by the VE. They can be tagged as public, low, medium, high or private according to the desired level of privacy. The public data are available as they are, either through the Message Bus (VE2COSMOS communication) or through VE2VE communication (GET or POST REST requests), while the private ones are not shared. For low, medium and high choices please see section 6.4.3.1.

6.4.3.1 Fuzzy Privelets

In many occasions (e.g. actual location) people want to share not their real data but (slightly) different ones, trying to reach a compromise between privacy and accuracy. Therefore, within the context of fuzzy Privelets, low, medium and high privacy level correspond respectively to small, medium or large (absolute) difference between the real value and the shared one.

Range

The Application Developer is responsible for defining properly the **range**, within which the shared value is calculated, in order for it to make sense. In the case of geospatial location, for example, suitable range must be selected for both the latitude and the longitude, to ensure that the shared location still belongs to the same neighborhood, town etc., according to the specific application.

Fuzzy Value Calculation

In order to calculate the shared value, Privelets component communicates with the H/W Random Generator to get a true random number. Then the component corresponds this number to the shared value exploiting some aspects of the fuzzy logic theory. The reason why we apply the fuzzy logic is the fact that people are unable to quantify their privacy preferences, but instead they are willing to describe them using linguistic variables such as “low”, “medium” or “high”. More analytically and trying to keep our model simple, we use the symmetric trapezoidal membership function to randomly map the user’s privacy preference with the absolute difference between the real value and the shared one:

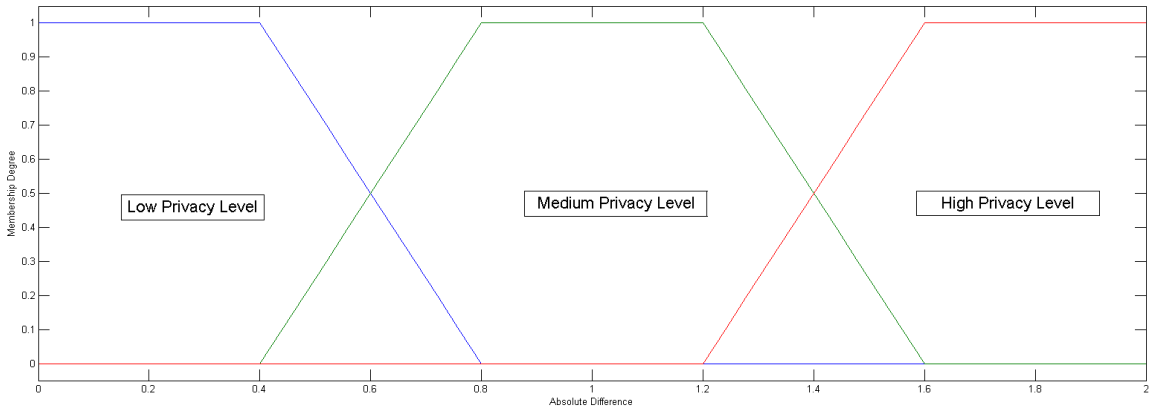


Figure 26: Symmetric Trapezoidal Membership Function

As an example, Figure 26 could depict the membership function which describes a data field (e.g. inside temperature in Celsius) whose real value is 20 and its range is defined as 10% (max Absolute Difference = 2).

Given that the Membership Degree (vertical axis) quantifies the grade of membership of the various Absolute Differences (horizontal axis) to each of the three fuzzy sets (low, medium,

high), we assigned larger possibilities to the differences where $MembershipDegree = 1$, through the following convention:

- Low Privacy Level:
  - The possibility that the absolute difference $\in (0, 0.2 * Range * \mathrm{Re}alValue) = 2/3$, because the $MembershipDegree = 1$
  - The possibility that the absolute difference $\in (0.2 * Range * \mathrm{Re}alValue, 0.4 * Range * \mathrm{Re}alValue) = 1/3$, because the $MembershipDegree \in (0,1)$
- Medium Privacy Level:
  - The possibility that the absolute difference $\in (0.2 * Range * \mathrm{Re}alValue, 0.4 * Range * \mathrm{Re}alValue) = 1/4$, because the $MembershipDegree \in (0,1)$
  - The possibility that the absolute difference $\in (0.4 * Range * \mathrm{Re}alValue, 0.6 * Range * \mathrm{Re}alValue) = 1/2$, because the $MembershipDegree = 1$
  - The possibility that the absolute difference $\in (0.6 * Range * \mathrm{Re}alValue, 0.8 * Range * \mathrm{Re}alValue) = 1/4$, because the $MembershipDegree \in (0,1)$
- High Privacy Level:
  - The possibility that the absolute difference $\in (0.6 * Range * \mathrm{Re}alValue, 0.8 * Range * \mathrm{Re}alValue) = 1/3$, because the $MembershipDegree \in (0,1)$
  - The possibility that the absolute difference $\in (0.8 * Range * \mathrm{Re}alValue, Range * \mathrm{Re}alValue) = 2/3$, because the $MembershipDegree = 1$

More details about the fuzzy value calculation can be found in D3.2.3.

6.4.4. Types of Communication

6.4.4.1 VE2COSMOS

The VE pushes its public and fuzzy data to the COSMOS Message Bus continuously. Assuming that the VE generates the following data

```
{
   "temperature":"21.4",
   "id":"abc123",
   "latitude":"440306",
   "longitude":"4474269",
   "domain":"flat",
   "ipAddress":"0.0.0.0"
}
```

Figure 27: VE JSON data

and that the configuration file is filled in like above, then a JSON message that could be published in the Bus is the following:


```
{
    "temperature":"21.1",
    "id":"abc123",
    "latitude":"443815",
    "longitude":"4469878",
    "domain":"flat",
}
```


Figure 28: VE JSON message

6.4.4.2 VE2VE

The following sequence describes the whole process that takes place when VE1 tries to access an IoT-Service of VE2 (e.g. “getTemp”):

- Find VE2: VE1 searches in its Followees’ list and retrieves the virtual IP address of VE2 and the request Key (please see **VE Recommendation phase** in section 6.4.2).
- Send the request: VE1 sends the request to VE2 with the request Key (e.g. http://10.0.0.2:3030/getTemp?requestKey=req1to2)
- Authentication at the side of VE2: VE2 searches in its Followers’ list and retrieves the request Key for the incoming virtual IP address (also the “hasLastTimeRequest” value, which is used in the next step, and the response Key) and tries to match it with the “req1to2”, which was received before. If they are equal, then the flow continues, else it stops because it seems that VE1 attempts to impersonate another VE (please see **VE Impersonation** paragraph below).
- Check for possible repetitive request: VE2 has already retrieved the “hasLastTimeRequest” value. If the current time minus this value is bigger than the “interval_between_requests” value, then the flow continues, else the request is ignored since it is considered as repetitive (annoying). The current time is stored as the new “hasLastTimeRequest” value.
- Check for the privacy level of the requested data: VE2 reads the configuration file and depending on the privacy level, it returns the real value, a fuzzy one, or nothing. In any case, the response contains the response Key “resp1to2”.
- Authentication at the side of VE1: VE1 searches in its Followees’ list and retrieves the response Key for the incoming virtual IP address and tries to match it with the “resp1to2”, which was received before. If they are not equal, then VE1 ignores the response, because it seems that VE2 attempts to impersonate another VE.

This process happens during all kinds of VE2VE communication (e.g. accessing a VE property, Experience Sharing etc.).

VE Impersonation

A VE can impersonate another VE by changing the “Virtual IP address” field in the VPN configuration file (please see **Configuration phase** in section 6.4.2) and entering the COSMOS VPN when the legitimate holder of the virtual IP address is offline. The impersonation can be detected either when interacting with other VEs or by the legitimate VE itself when attempting

to enter the VPN unsuccessfully (a conflict is caused when trying to use the same virtual IP Address). Every time a VE detects an impersonation, it automatically informs the privacy mechanism of the COSMOS platform using its PrivacyID. The mechanism validates the impersonation and provides the legitimate VE with a new virtual IP Address and also deactivates the old one. In addition, new request and response Keys, regarding the VE and its Followers, are generated.

6.4.5. Conclusions

VE Provider

When a VE acts as information provider, two are the critical points: the VE to be able to keep its identity private and also to protect all the data that are considered private. The first point is addressed by using virtual IP addresses (Year 2), whereas the second one by giving the VE Developer the capability to annotate the VE data in terms of the privacy level (Year 1). During Year 3 of the project, we enhanced the Privelets component with the fuzziness functionality, which can be applied in many real situations, where people are unwilling to share their exact information.

VE User

When a VE needs some kind of information, it communicates with the Social Monitoring component which ranks its Followees (possible information providers) according to their social indexes. These indexes mainly reflect the behaviour of the VEs when they act as providers (useful experiences, short response time, reliability, availability etc.) and are calculated based on users' feedback. Consequently the VE asks, in priority, the most socially dependable Followees, so our mechanism must ensure the (virtual) identity of them. This requirement is addressed using the response key which enables the VE user to validate the identity of the VE provider. Thus, Privelets aim to give added value to the Social Monitoring and Social Analysis component (for these components please see D5.2.3 [35] and D5.1.3 [36]).

COSMOS Platform

The platform is involved during the VE Recommendation phase and in case of VE impersonation, since the VEs themselves are responsible for authenticating each other when interacting. This reduces the whole traffic inside it and makes the mechanism much more decentralized.

7. Conclusions

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

In the first phase we set out to identify functional components of the COSMOS platform 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 first phase had two major goals. The first goal was to identify and mitigate security risks by enhancing COSMOS with dedicated hardware and software mechanisms. The second goal was to improve efficiency and speed by developing and extending already existing architectures leveraging 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 work performed in the three project years. WP3s tasks are split such that they cover all aspects of COSMOS – device level security, cloud level and data flow level security and consent management.

Using a top-down approach, the COSMOS Security Architecture is split into sub-components which are described in detail. Therefore a modular approach was used in order to facilitate the extension and mitigate redesign efforts. Security, performance and ease of were main drivers in the performed tasks. Starting at device level, WP3 has developed a security concept starting with the hardware. The root of trust, realized by means of hardware components such as cryptographic accelerators and true random number generators as well as low level software, built right from the concept level, enables high level software components to use a security anchor in order to provide services to end users.
In addition to COSMOS security, supporting auditing and consent management, audit-trails and privacy of tenants data at the storage layer are core components. WP3 provides a generic data auditing and policy enforcement to demonstrate the capability of supporting regulation for IoT related data. Also consent management is a key component of COSMOS which further enhances the platform by enabling end-users’ control over their data.

The tasks performed in WP3 were realized in collaboration with all other WP’s, with major inputs from WP2 and WP4. The COSMOS architecture, with respect to security, was realized together with WP2 resulting in a number of generic components which security enhance the platform. Together with WP4 the cloud aspects of security were tackled which has resulted in an extensible architecture covering everything from device up to cloud level.

The components developed in WP3, together with the architectural views can provide and end-to-end approach to IoT security by making use of current standards and regulations while allowing for customization and long term evolution.

8. 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 D2.3.2 Conceptual Model and Reference Architecture.

[20] AMBA specification retrieved feb. 2015: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0011a/index.html

[21] Diffie-Hellman 1, retrieved feb. 2015: http://wiki.openssl.org/index.php/Elliptic_Curve_Diffie_Hellman

[22] Diffie-Hellman 1, retrieved feb. 2015: http://en.wikipedia.org/wiki/Elliptic_curve_Diffie%E2%80%93Hellman

[23] Diffie-Hellman 1, retrieved feb. 2015: http://en.wikipedia.org/wiki/Elliptic_curve_cryptography

[24] NIST, Special Publication 800-56A, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, March, 2006. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf

[25] Extended Euclidian Algorithm, retrieved feb. 2015: http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

[26] "A Tutorial on Elliptic Curve Cryptography", Fuwen Liu, Brandenburg Technical University of Cottbus: http://vanilla47.com/PDFs/Cryptography/Miscellenea/Eliptic%20Curve%20Cryptography/A_tutorial_of_elliptic_curve_cryptography.pdf

[27] "ECC (Elliptic Curve Cryptography)", Francisco Rodríguez Henríquez, CINVESTAV: http://delta.cs.cinvestav.mx/~francisco/cripto/elliptic.pdf

[28] "SEC 2: Recommended Elliptic Curve Domain Parameters", Certicom Research, Sept. 2000. http://perso.univ-rennes1.fr/sylvain.duquesne/master/standards/sec2_final.pdf

[29] "Elliptic Curve Cryptography - An Implementation Guide", Anoop MS, Tata Elxi. http://www.infosecwriters.com/text_resources/pdf/Elliptic_Curve_AnnopMS.pdf

[30] "Theoretical Underpinnings of Modern Cryptography - Lecture 7: Finite Fields", Avi Kak Purdue University. https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture7.pdf

[31] "Finite Mathematics", Cambridge University. https://courses.cs.washington.edu/courses/cse466/11au/calendar/mackay-finite-field-appendix.pdf

[32] "Recommended Elliptic Curves For Federal Government Use", NIST, jul. 1999. http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf

[33] FreeLan: http://www.freelan.org/

[34] Apache Jena: https://jena.apache.org/

[35] COSMOS Project D5.2.3 Decentralized and Autonomous Things Management: Software prototype (Final)

[36] COSMOS Project D5.1.3 Decentralized and Autonomous Things Management: Design and open specification (Final

[37] Verizon Data Breach Investigation Report http://www.verizonenterprise.com/DBIR/

[38] OpeStack Keystone: http://docs.openstack.org/developer/keystone/

[39] Eclipse Paho project: http://www.eclipse.org/paho/

[40] Sima Nadler, Natalia Raznikov, Yosef Moatti, Shelly Garion, and Abigail Goldsteen, "A Data privacy and consent solution to enterprises", submitted.

[41] Sima Nadler, "It's Time to Radically Overhaul Approaches to Data Privacy", http://www.ibm.com/blogs/think/2016/01/27/its-time-to-radically-overhaul-approaches-to-data-privacy/

[42] "Retail Privacy Standards", http://ixtenso.com/en/story/30479-nrf-new-resources-to-help-retailers-protect-data.html

[43] Consent Management on Bluemix, http://consentmanagement.eu-gb.mybluemix.net/indexCosmos.html

[44] Consent Management API, http://consentmanagement.eu-gb.mybluemix.net/APIs/

[45] Panagiotis Bourelos, George Kousiouris, Orfefs Voutyras, Theodora A. Varvarigou, "Heating schedule management approach through decentralized knowledge diffusion in the context of social internet of things", Panhellenic Conference on Informatics, 2015, 103-108.

[46] Wikipedia, https://en.wikipedia.org/wiki/Privacy

[47] NIST random number genration, http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html

[48] Python Shell, https://github.com/extrabacon/python-shell