



Project no: 269317

## **nSHIELD**

new embedded Systems arcHitecturE for multi-Layer Dependable solutions

Instrument type: Collaborative Project, JTI-CP-ARTEMIS

Priority name: Embedded Systems

### **D3.2: Preliminary SPD Node Technologies Prototype**

Due date of deliverable: M18 – 2013.02.28

Actual submission date: M20 – 2013.04.23

Start date of project: 01/09/2011

Duration: 36 months

Organisation name of lead contractor for this deliverable:

Integrated Systems Development, ISD

Revision [Issue 8]

<b>Project co-funded by the European Commission within the Seventh Framework Programme (2007-2012)</b>		
<b>Dissemination Level</b>		
<b>PU</b>	Public	
<b>PP</b>	Restricted to other programme participants (including the Commission Services)	
<b>RE</b>	Restricted to a group specified by the consortium (including the Commission Services)	X
<b>CO</b>	Confidential, only for members of the consortium (including the Commission Services)	



## Document Authors and Approvals

Authors		Date	Signature
Name	Company		
Lorena de Celis	AT	18/02/2012	
Jacobo Domínguez	AT	18/02/2013	
David Abia	AT	18/02/2013	
Kyriakos Stefanidis	ATHENA	04/04/13	
Paolo Azzoni	ETH	15/03/13	
Stefano Gosetti	ETH	15/03/13	
George Dramitinos	ISD	04/02/13	
Cecilia Coveri	SES	14/03/13	
Antonio Di Marzo	SESM	05/02/13	
Antonio Bruscano	SESM	05/02/13	
Viktor Do	SICS	31/01/13	
Christian Gehrman	SICS	31/01/13	
Hans Thorsen	T2data	31/01/13	
Bharath siva kumar Tati	Telcred	06/02/13	
Carlo Pompili	Telcred	31/01/13	
Kostas Rantos	TUC	04/03/13	
Georgios Hatzivasilis	TUC	04/03/13	
Kostas Fysarakis	TUC	04/03/13	
Alexandros Papanikolaou	TUC	04/03/13	
Dimitris Geneiatakis	TUC	04/03/13	
Harry Manifavas	TUC	04/03/13	
Chiara Peretti	UNIGE	07/02/13	
Luca Noli	UNIGE	07/02/13	
Paolo Gastaldo	UNIGE	07/02/13	
Antonio Abramo	UNIUD	14/03/13	
Mirko Loghi	UNIUD	14/03/13	
Reviewed by			
Name	Company		
Approved by			
Name	Company		



## Applicable Documents

ID	Document	Description
[01]	TA	nSHIELD Technical Annex
[02]	D3.1	Deliverable D3.1: SPD Node Technologies Assessment
[03]	D2.4	Deliverable D2.4: Reference System Architecture Design
[04]	D2.5	Deliverable D2.5: Preliminary SPD Metric Specification

## Modification History

Issue	Date	Description
Issue 1	06/12/12	First version of ToC.
Issue 2	04/02/13	Integrated contributions from several partners.
Issue 3	11/02/13	Integrated contributions from several partners.
Issue 4	18/02/2013	Integrated contributions from several partners.
Issue 5	18/02/2013	Add AT contribution
Issue 6	05/03/13	Integrated contributions from partners
Issue 7	01/04/13	Integrated contributions from partners
Issue 8	16/04/13	First complete version of the deliverable.



## **Executive Summary**

This deliverable is focused on the detailed description of the node technologies that are currently under development in work package 3, conforming to the preliminary architecture and the composability requirements specified in deliverables D2.4 and D2.5. These technologies will be made available to the application scenarios and can be used as building blocks for the project demonstrators. This deliverable will be updated and refined in the second part of the project based on the final requests received from the application scenarios and on the refined system architecture, metrics and composition strategy to be followed.



## Contents

<b>1</b>	<b>Introduction .....</b>	<b>14</b>
<b>2</b>	<b>SDR/Cognitive Enabled Node Technologies .....</b>	<b>15</b>
<b>2.1</b>	<b>Hypervisor .....</b>	<b>15</b>
2.1.1	Virtualization .....	15
2.1.2	ARM Architecture.....	20
2.1.3	SICS Hypervisor .....	24
2.1.4	FreeRTOS Port.....	28
2.1.5	Linux Port.....	30
<b>2.2</b>	<b>Secure ES firmware .....</b>	<b>36</b>
2.2.1	Description.....	36
2.2.2	Architecture Modules .....	38
2.2.3	Architecture Interfaces.....	38
2.2.4	Boot Media Layout.....	39
2.2.5	Parameters Stored on the Motherboard.....	39
2.2.6	Metrics .....	40
<b>2.3</b>	<b>Smart Card Security Services in nSHIELD .....</b>	<b>40</b>
2.3.1	Building Secure Communications.....	40
2.3.2	Implementation .....	42
<b>2.4</b>	<b>Power Management &amp; Supply Protection Technology .....</b>	<b>42</b>
2.4.1	Description.....	42
2.4.2	Architecture Modules and Interfaces .....	44
2.4.3	Metrics .....	45
<b>2.5</b>	<b>MMC/SD Password Management .....</b>	<b>45</b>
<b>2.6</b>	<b>User Level Power Management.....</b>	<b>46</b>
<b>3</b>	<b>Micro/Personal Node .....</b>	<b>47</b>
<b>3.1</b>	<b>Face and Voice Recognition for People Identification .....</b>	<b>47</b>
3.1.1	Embedded System Based Face Recognition for People Identification.....	47
3.1.2	The Evaluation Framework.....	53
<b>3.2</b>	<b>Access Rights Delegation.....</b>	<b>60</b>
3.2.1	Problem Statement.....	61
3.2.2	The Concept of “Path Array” .....	61
3.2.3	Mechanism of the Artefact.....	62
<b>4</b>	<b>Power Node.....</b>	<b>65</b>
<b>4.1</b>	<b>Avionic System.....</b>	<b>65</b>
4.1.1	Description.....	65
4.1.2	Architecture Modules and Interfaces .....	65
4.1.3	Metrics .....	69
<b>4.2</b>	<b>Audio Based Surveillance.....</b>	<b>69</b>



	4.2.1	Description .....	69
	4.2.2	Architecture Modules and Interfaces .....	70
	4.2.3	Metrics .....	71
<b>4.3</b>		<b>System of Embedded System, SoES.....</b>	<b>71</b>
	4.3.1	Description .....	71
	4.3.2	Architecture Modules and Interfaces .....	72
	4.3.3	Metrics .....	74
<b>4.4</b>		<b>GPU accelerated hashing and hash lookup mechanism.....</b>	<b>75</b>
	4.4.1	General description .....	75
	4.4.2	Development & performance comparisons .....	77
<b>5</b>		<b>Dependable self-x Technologies.....</b>	<b>79</b>
<b>5.1</b>		<b>Platform Selection.....</b>	<b>79</b>
	5.1.1	Metrics .....	85
	5.1.2	Demo.....	85
<b>5.2</b>		<b>Anonymity Service.....</b>	<b>87</b>
	5.2.1	Network Topology & Connectivity .....	88
	5.2.2	Implementation Details.....	91
	5.2.3	Node Communication.....	93
	5.2.4	Server Platform .....	93
	5.2.5	Anonymity Scheme Demonstration .....	94
<b>5.3</b>		<b>DDoS Attack Mitigation on SPD Power/Micro Nodes.....</b>	<b>98</b>
	5.3.1	Introduction.....	98
	5.3.2	Packet Marking Schemes .....	100
	5.3.3	Filtering and Traceback Mechanism .....	102
	5.3.4	Conclusions.....	107
<b>6</b>		<b>Cryptographic technologies.....</b>	<b>108</b>
<b>6.1</b>		<b>Elliptic Curve Point Multiplication over Prime Fields Library.....</b>	<b>108</b>
	6.1.1	Description .....	108
	6.1.2	Basic Interface.....	108
	6.1.3	Code Compilation and Dependencies .....	108
	6.1.4	Hardware platform.....	109
	6.1.5	Library API.....	109
	6.1.6	Service functions .....	109
	6.1.7	Code Samples.....	110
<b>6.2</b>		<b>Compact Crypto Library .....</b>	<b>110</b>
	6.2.1	Description .....	110
	6.2.2	Basic Interface.....	110
	6.2.3	Code compilation and Dependencies .....	110
	6.2.4	Hardware platform.....	111
	6.2.5	Library API.....	111
	6.2.6	Service Functions.....	111
	6.2.7	Code Samples.....	112
<b>6.3</b>		<b>Anti-tamper technologies.....</b>	<b>113</b>
	6.3.1	Encapsulation and physical barriers .....	113



6.3.2	Supervisor chips .....	115
6.3.3	Modules interfaces.....	115
6.3.4	Metrics .....	116
<b>6.4</b>	<b>An Identity-Based Encryption scheme .....</b>	<b>116</b>
6.4.1	Description.....	116
6.4.2	Hardware it will be deployed on.....	117
6.4.3	Implementation and Security Issues .....	118
<b>6.5</b>	<b>Secure Cryptographic Key Exchange using the Controlled Randomness Protocol.....</b>	<b>118</b>
6.5.1	Introduction .....	118
6.5.2	The Controlled Randomness Protocol .....	119
6.5.3	Methodology .....	121
6.5.4	Results and Discussion .....	121
6.5.5	Conclusions .....	122
<b>7</b>	<b>References.....</b>	<b>123</b>



## Figures

Figure 2-1: Architecture of a hypervisor system.....	16
Figure 2-2: Page table fetch. ....	23
Figure 2-3: MMU Domains .....	25
Figure 2-4: Kernel mode domain access .....	25
Figure 2-5: Task mode domain access .....	26
Figure 2-6: Trusted mode domain access.....	26
Figure 2-7: FreeRTOS hypervisor system .....	29
Figure 2-8: System memory layout .....	33
Figure 2-9: Secloader functionality.....	37
Figure 2-10: Load image format.....	39
Figure 2-11: High level components of the proposed implementation.....	42
Figure 2-12: Smart power unit.....	43
Figure 2-13: nSHIELD Reference architecture – Node Layer (power module) .....	45
Figure 3-1: Recognition process main phases.....	48
Figure 3-2: Space distribution of faces images.....	49
Figure 3-3: Example of a simple "face space" consisting of just two eigenfaces (u1 and u2 ) and from three individuals known ( $\Omega_1$ , $\Omega_2$ and $\Omega_3$ )......	51
Figure 3-4: Eigenface in which domains were identified: eigeneye (left and right), eigennose and eigenmouth. ....	52
Figure 3-5: Example of identification of eigenfeature.....	52
Figure 3-6: The evaluation framework in the recognition system life cycle.....	53
Figure 3-7: The testing protocol. ....	55
Figure 3-8: Images from one subject session. (a) Four controlled stills, (b) two uncontrolled stills, and (c) 3D shape channel and texture channel pasted on 3D shape channel. ....	56
Figure 3-9: Demographics of FRP ver2.0 validation partition by (a) race, (b) age, and (c) sex.....	57
Figure 3-10: Histogram of the distribution of subjects for a given number of replicate subject sessions. The histogram is for the ver2.0 validation partition. ....	57
Figure 3-11: Example of expected baseline ROC performance for test 1, 2, 3, and 4. ....	58
Figure 3-12: Example of baseline ROC performance for Experiment 3 component study. ....	59





Figure 3-13: Estimated densities. ....	60
Figure 3-14: Path Array Design .....	61
Figure 3-15: Ticket along with Path Array.....	61
Figure 3-16: Ticket Incrementing the index value.....	62
Figure 3-17: Process of HMAC creation .....	63
Figure 4-1: OMNIA Architecture. ....	66
Figure 4-2: RIU HW block diagram.....	67
Figure 4-3: APM460SW.....	68
Figure 4-4: The audio daughterboard .....	70
Figure 4-5: The grabber board.....	70
Figure 4-6: nS-ESD GW Integration .....	72
Figure 4-7: IP Logical Architecture. ....	72
Figure 4-8: Hardware Accelerator.....	74
Figure 4-9: Functionality sketch of GPU-accelerated hashing mechanism.....	76
Figure 4-10: Basic file check utilizing the SHA-3 hash function .....	76
Figure 4-11: Data structure performance comparison for CUDA-accelerated implementation Dependable self-x Technologies .....	77
Figure 4-12: Comparison of execution and I/O time of SHA-3 implementations.....	78
Figure 5-1: Prototype schematic.....	79
Figure 5-2: The OMBRA board.....	81
Figure 5-3: Demo for 256 bit curve .....	86
Figure 5-4: Demo for 521 bit curve .....	87
Figure 5-5: Overlay of anonymity service node topology over floor layout.....	88
Figure 5-6: The anonymity server and its connection to nodes.....	88
Figure 5-7: The Sensor "C" moving to new location.....	89
Figure 5-8: Service topology including users.....	89
Figure 5-9: Example of cloaked area calculation.....	90
Figure 5-10: Node data structure.....	91
Figure 5-11: Peer structure .....	91
Figure 5-12: Peer list structure. ....	92



Figure 5-13: Neighbourhood and Neighbour's cloaked area structures. ....	92
Figure 5-14: Area and Pair structures. ....	92
Figure 5-15: Typical node message exchange. ....	93
Figure 5-16: Server application. Cloaked area received from node "6". ....	94
Figure 5-17: Typical Initialization of node "6". ....	94
Figure 5-18: Typical Initialization of node "4". ....	95
Figure 5-19: Node "6" running the anonymity algorithm. ....	96
Figure 5-20: Node "4" running the algorithm. ....	96
Figure 5-21: Node "6" achieved the desire K-anonymity level. ....	97
Figure 5-22: "4" reached the required k-Anonymity level. ....	97
Figure 5-23: Node "6" cannot find K users, thus node "4" sends its peer list to node "6". ....	98
Figure 5-24: Typical error message. ....	98
Figure 5-25: The IP header in packet marking I. In packet marking scheme I, we overload the shaded part of the IP header to put the 17 bit packet marking. ....	101
Figure 5-26: The false positives of packet marking scheme I for 130 to 150 thousand edge routers. As the attack sources rise, we notice a considerable decline in false positives. ....	101
Figure 5-27: The IP header in packet marking II. In packet marking scheme II, we overload the shaded part of the IP header to put the 32 bit packet marking. ....	102
Figure 5-28: The filtering and traceback mechanism architecture. ....	103
Figure 5-29: Test case 1. A DDoS attack starts progressively. The control centre responds and stabilizes the server load to a comfortable level. ....	106
Figure 5-30: Test case 2. This DDoS attack incorporates networks that are of value to the victim's organization. Some networks stop sending traffic in the middle of the attack while others join the attack later. ....	107
Figure 6-1 – Secure encapsulated module ....	114
Figure 6-2 – Secure PCMCIA Card.....	114
Figure 6-3 – Secure plug-on module.....	114
Figure 6-4 – Tamper respondent surface enclosure .....	115
Figure 6-5: Reference architecture - Node layer (anti-tamper component) .....	116
Figure 6-6: Key generation using the Sakai-Ohgishi-Kasahara IBE protocol .....	118



## Tables

Table 2-1: Page table AP Configuration .....	23
Table 2-2: Hypercall interface .....	27
Table 2-3: Kernel Memory Layout on ARM Linux.....	31
Table 2-4: Page table access permissions configuration .....	33
Table 2-5: Domain access configuration .....	33
Table 2-6: Integrator CP platform IO .....	35
Table 2-7: Parameter memory contents .....	40
Table 2-8: Power module at node layer.....	45
Table 3-1: Size of faces in the validation set imagery broken out by category. Size is measured in pixels between the centres of the eyes. The table reports mean, median, and standard deviation. ....	56
Table 5-1: Prototype features .....	80
Table 5-2: Each record in the repository denotes a network and has a set of standard fields (those marked with *) and user specific fields along with their respective weights.....	105
Table 6-1: Reference architecture – Node Layer (anti-tamper attributes).....	116



## **Glossary**

Please refer to the Glossary document, which is common for all the deliverables in nSHIELD.



*This page is intentionally left blank*

# 1 Introduction

The nSHIELD project proposes a layered architecture to provide intrinsic SPD features and functionalities to embedded systems. In this layered architecture work package 3 is responsible for the node layer that represents the lower level of the architecture, a basement constituted of real embedded devices on which the entire project will grow.

As already outlined in the TA, work package 3 aims to create an Intelligent ES HW/SW Platform that consists of three different kinds of Intelligent ES Nodes: nano node, micro/personal node and power node. These three categories of embedded systems will represent the basic components of the lower part of an SPD Pervasive System that will cover the possible requirements of several market areas: from field data acquisition, to transportation, to personal space, to home environment, to public infrastructures, etc.

The assessment of the state of the art as well as of the technologies to be developed in the context of the project is contained in deliverable D3.1. This deliverable is focused on the detailed description of the node technologies that are currently under development in work package 3, conforming to the preliminary architecture and the composability requirements specified in deliverables D2.4 and D2.5. These technologies will be made available to the application scenarios and can be utilized as building blocks for the project demonstrators. This deliverable will be updated and refined in the second part of the project based on the final requests received from the application scenarios and on the refined system architecture, metrics and composition strategy to be followed.

The document is structured in the following sections:

1. Introduction: a brief introduction.
2. SDR/Cognitive Enabled node: SDR/Cognitive Enabled Node (CEN) technologies for generic application scenarios including technologies for secure booting, isolation of critical security tasks and power management.
3. Micro/Personal node: Technologies required by scenarios 2 (Voice/Facial Recognition) and 4 (Social Mobility) at node level. It focuses mainly on biometric algorithms for SPD and on the delegation of access rights for offline control systems.
4. Power node: this section describes the technologies that will be adopted in the areas of surveillance, system of embedded systems and SPD for avionics. These technologies will be adopted in scenarios 1 (Railways security), 3 (Dependable Avionic Systems) and 4 (Social Mobility).
5. Dependable self-x Technologies: this section introduces horizontal SPD technologies that will be adopted in task 3.1-3.2-3.3 at different levels, depending on the complexity of the node and considering its HW/SW capabilities, its requirements and its usage. The technologies are focused on areas such as denial-of-services and anonymity.
6. Cryptographic technologies: this section provides the assessment of horizontal SPD technologies focused specifically on hardware and software cryptography, on the use of crypto technologies to implement SPD embedded devices and prevent physical attacks at this level using defence crypto-based solutions.

## 2 SDR/Cognitive Enabled Node Technologies

The main research areas targeted by the SDR/Cognitive enabled node technologies are the following:

- Intrinsically secure ES firmware including:
  - A hypervisor for ARM that allows security critical applications to run isolated, co-existing in the same system with less trustworthy or even insecure applications.
  - A secure boot loader for ARM ensuring the integrity of the images to be loaded.
  - Provision of confidentiality, integrity or/and authenticity services to nodes using smartcards.
- Power management and supply protection, including:
  - A family of Smart Power Units that is flexible enough to be used as a potential solution for power supply management a protection by the application scenarios.
  - A Linux kernel module supporting password protection for MMC/SD cards to avoid power consumption resulting from encrypting data that are not considered to be too critical.
  - A user level interface exposing allowing applications to tune their computational requirements and their power consumption and an activity profiler that allows collecting information about the whole system behaviour enabling to control the power consumption by selecting the most effective energy policy as function of the computational load and of the power supply status.

### 2.1 Hypervisor

The technological advancement in embedded systems has led to new possibilities to run open and complex operating systems, in which before was privileged to personal computers. As these systems become more interconnected across networks and the Internet, there is a clear indication of threats increasing, targeting mobile and sensitive infrastructure devices [1]. This is indeed true for all nSHIELD nodes, and to combat the associated risks, SICS has developed a Hypervisor to provide a secure execution environment that allows trustworthy, security critical applications to run isolated, co-existing in the same system with less trustworthy or even insecure applications. In the following chapters, we will describe the fundamental technology in which the Hypervisor is built on, and how it can provide isolation for security critical applications.

#### 2.1.1 Virtualization

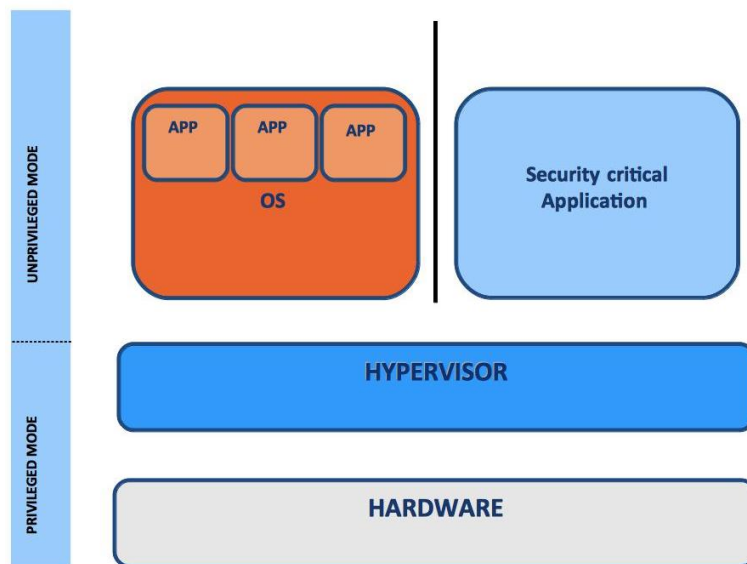
In computer science, the term virtualization can refer to many things. Software can be virtual, as can memory, storage, data and networks. In this section, virtualization refers to system virtualization in which a piece of software, the hypervisor also known as a virtual machine monitor (VMM), runs on top of the physical hardware to share the hardware's full set of resources between its guests called virtual machines (VM). Virtualization is normally associated as a means to increase utilization, server consolidation and cross platform interoperability in enterprise systems and desktop computers. However in this case, for embedded systems, we are more interested in enhancing security by isolating different execution environments by having low level barebone security applications running isolated co-existing with a high level rich operating system. Contrary to desktop computers, the configuration can be quite static, rather than the dynamic creation and destruction of OS environments.

##### 2.1.1.1 Hardware Support for Virtualization

CPU architectures provide multiple operational modes, each with a different level of privilege. Most current ARM architectures have two modes, privileged and unprivileged mode, while the new Cortex-A15 have more extended virtualization support with three modes [2]. The CPU architectures we primarily target only have two operational modes. These different modes enforce the security of the system's resources and execution of privileged instructions.

Generally operating systems are designed to run directly on top of hardware and expect to run in privileged mode in order to take total control over the whole computer system. However in a virtualized environment, the hypervisor will be running in privileged mode while the operating system is modified to run in unprivileged mode. This complicates matters, as the operating system will not be able to execute the privileged instructions necessary to configure and drive the hardware directly. Instead, the privileged instructions need to be delegated to the hypervisor in order to provide the hardware safely to the guest VM. Another complication is that the user applications and the OS kernel now runs in the same privileged mode, which means that we have to introduce new virtual guest modes (virtual user mode and virtual kernel mode) in order to keep the kernel and user application isolated from each other.

The figure below describes the hypervisor architecture. We have the hypervisor running in the most privileged mode right above the hardware. The guest OS together with its applications are in turn running on top of the hypervisor in a less privileged mode. There is also a security critical application running on top of the hypervisor isolated from the other OS. These can be seen as guest VM's of the hypervisor. The hypervisor thus maintains the resource allocation of the guests, while it also has the power to intercept on important instructions and events and handle them before they are executed on the real hardware. By utilizing the different operating modes of the CPU (supervisor/user), the memory management unit (MMU) and the different domains, the hypervisor can setup an access policy that satisfies the security and isolation of the embedded system. The guest OS can only communicate with the security critical application through well-defined interfaces that the hypervisor provides.



**Figure 2-1: Architecture of a hypervisor system.**

The following is the main advantage of virtualization for embedded systems

- Isolation
- Minimized size of trusted code base (TCB)
- Improved security

In the next section we will describe the different virtualization approaches that can be used to implement a hypervisor.

### 2.1.1.2 Classical Virtualization

Popek and Goldberg stated in their paper, [3], formal requirements for computer architectures to be virtualizable. The classifications of sensitive and privileged instructions were introduced in their paper:



- Sensitive instructions, instructions that attempt to interrogate or modify the configuration of resources in the system.
- Privileged instructions, instructions that trap if executed in an unprivileged mode, but execute without trapping when run in a privileged mode.

To be able to fully virtualize architecture, Popek and Goldberg stated that the sensitive processor instructions had to be equal to the set of privileged instructions or a subset of it. This criterion has now been termed classically virtualizable. In the following section we present different types of virtualization techniques as each has its own advantages and disadvantages.

### 2.1.1.3 Full Virtualization

As discussed earlier, because the hypervisor resides in the privileged mode, the guest OS which is residing in the less privileged mode, cannot execute its privileged instructions. Instead the execution of these privileged instructions has to be delegated to the hypervisor. One way to do this is through applying full virtualization. The idea behind it is, whenever software is trying to execute privileged instructions in an unprivileged mode, it will generate a trap into the privileged mode. Because the hypervisor resides in the privileged mode, one could write a trap handler that emulates the privileged instruction that the guest OS is trying to execute. This way, through trap-and-emulate, all the privileged instructions that the guest OS is trying to execute will be handled by the hypervisor, while all other non-privileged instructions can be run directly on the processor. The advantage with full virtualization is that the virtualized interfaces provided to the guest operating system has identical interfaces compared to the real machine. This means that the system can execute binary code without any changes, neither the operating systems nor their applications need any adaptation to the virtual machine environment and all code that had originally been written to the physical machine can be reused.

However to apply full virtualization it requires that all sensitive instructions are a subset of the privileged instructions, in order for it to trap to the hypervisor. This is why Popek and Goldberg's criteria classically virtualizable have to be fulfilled in order to apply full virtualization and most general purpose CPU's does not support this. A downside with full virtualization is, since a trap is generated for every privileged instruction, it adds significant overhead as each privileged instruction is emulated with many more instructions. In turn we get excellent compatibility and portability.

### 2.1.1.4 Binary Translation

In the 90s, x86 architecture was prevalent in desktop and server computers but still, full virtualization could not be applied to the architecture. Because the x86 architecture contains sensitive instructions that are not a subset of the privileged instructions [4], it fails to fulfil Popek and Goldberg's criteria "classically virtualizable". These sensitive instructions would not trap to the hypervisor and it was not possible to execute these sensitive instructions in the unprivileged mode, making full virtualization not possible. VMware has however shown that, with binary translation one could also achieve the same benefits as full virtualization on x86 architecture. Binary translation solves this problem by scanning the guest code at load or runtime for all sensitive instructions that do not trap before they are executed, and replaces them with appropriate calls to the hypervisor. The technique used is quite complex and increases the code size running in the highest privileged mode, increasing the TCB. Through a security point of view, one would want the amount of code in the privileged mode to be as small as possible in order to minimize the area of the attack surface. This could affect the security and isolation properties of the entire system.

Because of the complex scanning techniques of binary translation, the performance overhead is larger than full virtualization. However, binary translation has provided the benefits of full virtualization on an architecture that was previously not fully virtualizable.

### 2.1.1.5 Paravirtualization

Para-virtualization was designed to keep the protection and isolation found in the full virtualization but without the performance overheads and implementation complexity in the hypervisor. However, in order to achieve this, you have to sacrifice the convenience of running an operative system unmodified on the hypervisor.

In a paravirtualized system, all the privileged instructions in the operating system kernel have to be modified to issue the appropriate system call that communicates directly with the hypervisor, also called hypercalls. This makes paravirtualization able to achieve better performance compared to full virtualization due to the direct use of appropriate hypercalls instead of multiple traps and instruction decoding. Examples on hypercall interfaces provided by the hypervisor are critical kernel operations such as memory management, interrupt handling, kernel ticks and context switching. As each hypercall offers a higher level of abstraction compared to emulation at the machine instruction level, the amount of work that a hypercall can do is a lot more efficient compared to emulating each sensitive machine instruction.

Most ARM architectures<sup>1</sup>, which are very common in embedded systems, however do not fulfil the criteria of "classically virtualizable". This means that virtualization on the ARM architecture can either be achieved through binary translation or paravirtualization. Because embedded systems generally are resource constrained, the performance overhead that binary translation generates is too high, making paravirtualization the best approach for the ARM architecture.

However, the drawback with paravirtualization is that each operating system has to be adapted to the new interface of the hypervisor. This can be quite a large task, and closed-source operating systems like Windows cannot be modified by anyone other than the original vendor. Still, in embedded systems it is common for the developers to have full access to the operating system's source code. The disadvantage to run a modified operating system is not always a big issue; the operating system needs to be ported to the custom hardware either way and at the same time, it performs better.

#### 2.1.1.6 Hardware Virtualized Extensions ARM

Focusing on the ARM architecture, it is worth to mention that it offers a hardware security extension called TrustZone [5] in ARMv6 or later architectures. It offers support for switching between two separate states, called worlds. One world is secure which is intended to run trusted software, while the other world is normal, where the untrusted software runs.

A single core is able to execute code from both worlds, and at the same time ensuring that the secure world software is protected from the normal world. Thus, the secure world controls all partitioning of devices, interrupts and coprocessor access. To control the switch between the secure and normal world, a new processor mode has been introduced called Monitor mode, preventing data from leaking from the secure world to the normal world.

In the latest ARMv7 architecture, the Cortex-A15 processor further introduced hardware virtualization extensions that allow the architecture to be classically virtualized by bringing a new mode called *hyp* as the highest privileged mode, hardware support for handling virtualized interrupts, and extra functionality to support and simplify virtualization. These extra extensions add features to make full virtualization possible and improve the speed of virtualization [2].

#### 2.1.1.7 Virtualization in Embedded Systems

As the nSHIELD project focuses in embedded systems, we will look into the functionality of virtualization that is inherited from their previous use in servers and workstations. The properties between the two systems are however completely different. For server and desktop computers, power, space or weight is of no concern, while for embedded systems the contrary often holds true. So a re-evaluation in the light of embedded systems is necessary.

Because the server and desktop markets are largely dominated by x86 architecture, virtualization approaches have been specifically tailored for this architecture. Also for server and desktops, usually the number one requirement is to be able to run all commodity operating systems without modifications. This was the advantage that full virtualization had over paravirtualization, but in embedded systems, it is common for the developer to have access to the full source code of the operating system. Usually the

---

<sup>1</sup> With the exception of the new ARMv7 Cortex-A15

developers have to port the operating system to the specialized embedded hardware, thus using paravirtualization is not such big disadvantage anymore.

#### **2.1.1.8 Isolation**

In servers and desktops, all virtualization approaches featured strong isolation between the VM's and is usually all that is needed to provide a secure and robust environment. A VM that is affected by malicious software will be confined to that VM, as the isolation prevents it from spreading to other VM's. For server and desktop use, this is usually all that is needed because there is no need for VM's to interact with each other in any other ways from how real computers communicate, that is through the network. However in embedded systems, multiple systems generally contribute to the overall function of the device. Thus the hypervisor needs to provide a secure communication interface between the isolated VM's, much like a microkernel IPC, while still preserving the isolation of the system [6].

#### **2.1.1.9 Trusted Computing Base**

In embedded systems, the size of the memory has a big effect on the cost of the device. They are generally designed to provide their functionality with minimal resources, thus cost and power sensitive devices benefits from a small code size.

In other devices where the highest levels of safety or security are required, every code line represents an additional potential threat and cost. This is called the trusted code base and includes all software that is run in privileged mode, which in general cases include the kernel and any software modules that the kernel relies on. In security critical applications, all trusted code may have to go through extensive testing. In some cases where security needs to be guaranteed, the security of the system has to be proven mathematically correct and undergo a formal verification. This makes it crucial that the size of the trusted code base is as small as possible as it will make formal verification easier.

In virtualization, the trusted code base will include the hypervisor as it now runs in the most privileged mode. For data server hypervisors like Xen [7], its code base is about 100,000 lines of code which is quite large, but the biggest problem is that it also relies on a full Linux system in the privileged mode. This makes the trusted code base several millions lines of code which makes a formal verification impossible. The reason the Xen and similar hypervisors is so large, is because it is mainly designed for server stations. Most policies are implemented inside the privileged code, which embedded systems have very little, or no use of.

In a microkernel the servers provide all the policies, while the microkernel only provides the mechanism to execute these policies. It aims to reduce the amount of privileged code to a minimum but still provide the basic mechanism to run an operating system on it. This result in a small trusted code base and from a security perspective, for example the L4 microkernel has a big advantage as the size is only about 10,000 lines of code and has also undergone formal verification [8].

#### **2.1.1.10 Performance**

Most often performance is much more crucial and expensive for embedded systems. To be able to get the most out of the hardware, a hypervisor for embedded systems must perform with a very low overhead as well as being able to provide good security and isolation. The performance overhead that the hypervisor generate depends on many factors such as the guest operating system, hypervisor design and hardware support for virtualization. For embedded systems, it is almost always advantageous to apply paravirtualization as a hypervisor design approach, for the reasons stated in section 2.1.1.7.

#### **2.1.1.11 Virtualization Summary**

Awareness that embedded systems also can benefit from virtualization as a mean to improve security; efficiency and reliability have increased the popularity of embedded virtualization. As the performance of embedded systems continues to grow, one single embedded system is now powerful enough to handle workloads, which previously had to be handles by several dedicated embedded systems. As they also become increasingly interconnected through networks and the Internet, security issues and malicious software has become a threat even in small-embedded nodes.

One question one might ask is why implement virtualization when one can use hardware virtualized extensions like TrustZone on ARM. One main point is that additional hardware support is more expensive as it is only available on a limited set of the ARM family CPUs, additionally it requires that the SoC design is made according to the TrustZone principles from the very beginning in order to work, as sensitive units such as DMA, memory interfaces, and interrupt controllers must be designed to support TrustZone. Hence, if we in principle can fulfil the same security requirements using only the MMU and standard processor protection mechanism, why not use such a design instead as long as the performance impacts are reasonable. Furthermore, the hypervisor is more flexible solution as it can be configured to work for a specific security policy and has the possibility to interpose and monitor all the important events in the "Normal world", something that TrustZone does not allow the user [9]. Although the hypervisor approach requires substantially larger porting effort it is still a very attractive means to provide isolation between different execution environments, separating your security critical applications from the rest.

## 2.1.2 ARM Architecture

In order to understand how virtualization is implemented in the ARM architecture, we provide an overview over important components on the ARMv7-A architecture. More detailed information can be found in the ARM reference manual [10].

### 2.1.2.1 Introduction

The ARM core is RISC architecture. RISC philosophy concentrates on reducing the complexity of instructions performed by the hardware, while putting a greater demand on the compiler. This way, each instruction is of fixed length of 32-bits and can be completed in a single clock cycle, while also allowing the pipeline to fetch future instruction before decoding the current instruction.

In contrast to RISC, CISC architectures relies more on hardware for instruction functionality, which consequently makes the instructions more complex. The instructions are often variable in size and take many cycles to execute.

As a pure RISC processor is designed for high performance, the ARM architecture uses a modified RISC design philosophy that also targets code density and low power consumption. As a result, the processor has become dominant in mobile embedded systems. It was reported that in 2005, about 98% of more than a billion mobile phones sold each year used at least one ARM processor and as of 2009, ARM processors accounted for approximately 90% of all embedded 32-bit RISC processors [11].

### 2.1.2.2 Current Program Status Register

Beside the 16 general-purpose registers from r0 to r15 in the ARM architecture, we have the CPSR, which the ARM processor uses to monitor, and control internal operations. The CPSR is used to configure the following:

- **Processor mode:** Can be in seven different processor modes, discussed in the next section.
- **Processor state:** The processor state determines if ARM, Thumb or the Jazelle instruction set is being used<sup>2</sup>.
- **Interrupt masks:** Interrupt masks are used to enable or disable the FIQ and IRQ interrupts.
- **Condition flags:** Contains the results of ALU operations, which update the CPSR condition flags. These are instructions that specify the S<sup>3</sup> instruction suffix and are used for conditional execution to speed up performance.

---

<sup>2</sup> ARM - 32 bit, Thumb - 16-bit, Jazelle - 8 bit (Java byte code support)

<sup>3</sup> Certain instructions have the possibility to add an optional S suffix to the instruction

### 2.1.2.3 Processor mode

The ARM architecture contains seven processor modes, which are either privileged or unprivileged. It contains one unprivileged mode User and the following modes are all privileged.

- Supervisor
- Fast interrupt request (FIQ)
- Interrupt request (IRQ)
- Abort
- Undefined
- System

When power is applied to the processor, it starts in Supervisor mode and is generally also the mode that the operating system operates in. FIQ and IRQ correspond to the two interrupt levels available on the ARM architecture. When there is a failed attempt to access memory, the processor switches to abort mode. System mode is generally used for other privileged OS kernel operations. Undefined mode is used when the processor encounters an instruction that is undefined or unsupported by the implementation. Lastly, the unprivileged mode User mode is generally used for programs and applications run on the operating system. In order to have full read/write access to the CPSR, the processor has to be in privileged mode.

### 2.1.2.4 Interrupts and Exceptions

Whenever an exception or interrupt occurs, the processor suspends the ongoing execution and jumps into the corresponding exception handler in the vector table. The vector table is located in a specific memory address and each 4-byte entry in the table contains an address, which points to the start of a specific routine:

- **Reset:** Location of the first instruction executed by the processor at power up. The reset vector branches to the initialization code.
- **Undefined:** When the processor cannot decode an instruction, it branches to the undefined vector. Also occurs when a privileged instruction is executed from the unprivileged user mode.
- **Software interrupt:** Occurs when the SWI instruction is used. The instruction is frequently used by applications when invoking an operating system routine. When used, the processor will switch from user mode to supervisor mode.
- **Prefetch abort:** Occurs when the processor trying to fetch an instruction from an address without the correct access permissions.
- **Data abort:** Occurs when the processor attempts to access data memory without correct access permissions.
- **Interrupt request:** Used by external hardware to interrupt the normal execution flow of the processor.

What the specific routine will do is generally controlled by the operative system. However, when applying virtualization to the system, all the routines will be implemented inside the hypervisor.

### 2.1.2.5 Coprocessor

The ARM architecture makes it possible to extend the instruction set by adding up to 16 coprocessors to the processor core. This makes it possible to add more support for the processor, such as floating-point operations.

Coprocessor 15 is however reserved for control functions such as the cache, memory management unit (MMU) and the translation looks aside buffer (TLB). In order to understand how the hypervisor can

provide improved security by isolating different resources, it is important to understand the mechanics behind the memory management of the ARM architecture.

### 2.1.2.6 Memory Management Unit

Through coprocessor 15 on the ARM architecture, the MMU can be enabled. Without an MMU, when the CPU accesses memory, the actual memory addresses never change and map one-to-one to the same physical address. However with an MMU, programs and data are run in virtual memory, an additional memory space that is independent of the physical memory. This means that the virtual memory addresses have to go through a translation step prior each memory access. It would be quite inefficient to individually map the virtual to physical translation for every single byte in memory, so instead the MMU divides the memory into contiguous sections called pages. The mappings of the virtual addresses to physical addresses are then stored in the page table. In addition to this, access permission to the page table is also configurable.

To make the translation more efficient, a dedicated hardware, the TLB handles the translation between virtual and physical addresses and contains a cache of recently accessed mappings. When a translation is needed, the TLB is searched first, and if not found, a page walk occurs which means it continues to search through the page table. When found, it will be inserted into the TLB, possibly evicting an old entry if the cache is full.

The virtualization of memory efficiently supports multitasking environments as the translation process allows the same virtual address to be held in different locations in the physical memory. By activating different page tables during a context switch, it is possible to run multiple tasks that have overlapping virtual addresses. This approach allows all tasks to remain in physical memory and still be available immediately when a context switch occurs.

### 2.1.2.7 Page Tables

The page table descriptors are architecture specific, and we will show how the ARMv7-A architecture looks like.

There are two levels of page tables in the ARM MMU hardware. The master page table contains 4096 page table entries known as first level descriptors, each describing 1MB of virtual memory, enabling up to 4GB of virtual memory. The level one master page table can either be a super section descriptor, section descriptor or page table descriptor.

- **Supersection descriptor:** Consists of 16MB block of memory. Because each first level page table entry covers a 1MB region of virtual memory, the 16MB supersections require that 16 identical copies of the first level descriptor of the supersection exist in the first level page table.
- **Section descriptor:** Consists of 1MB block of memory
- **Page descriptor:** Provides the base address to a second level descriptor that specifies how the associated 1MB is mapped. It's either a large or small page descriptor.

Second level descriptors:

- **Large page descriptor:** Consist of 64KB blocks of memory
- **Small page descriptor:** Consist of 4KB blocks of memory

The large page descriptor thus has 64 entries while a small page has 256 entries, splitting the 1MB that the table describes into 64KB and 4KB of blocks respectively. The figure below shows the overview of the first and second level page tables.

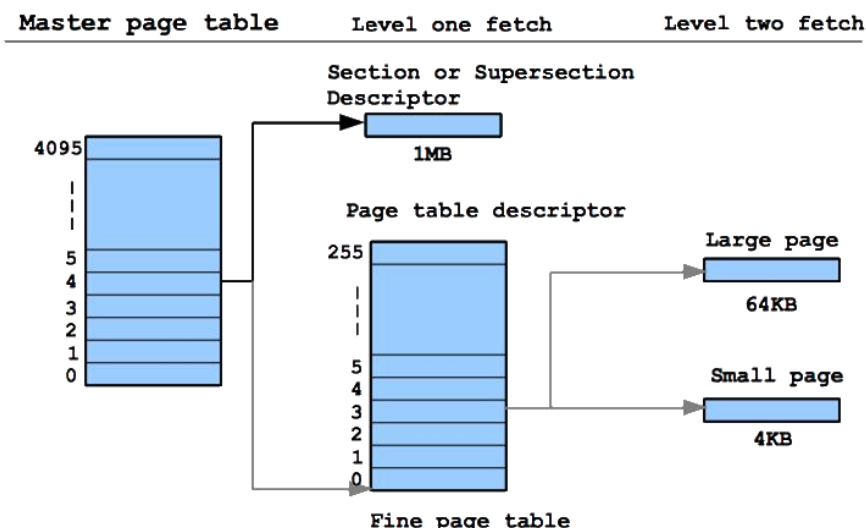


Figure 2-2: Page table fetch.

The translation process always begins in the same way at system start-up; the TLB does not contain a translation for the requested virtual address so it initiates a level one fetch. If the address is a supersection or section-mapped access it returns the physical address and the translation is finished. But if it is a page-mapped access, it requires an additional level two fetch into either a large or small page in where the TLB can extract the physical address.

Common for all levels of page tables is that it contains configuration for cache, buffer and access permission. The domain configuration is however only configurable for the first level descriptors, associating the page table with one of the 16 MMU domains. This means that it can only be applied at 1MB granularity; individual pages cannot be assigned to specific domains.

2.1.2.8 Domain and Memory Access Permissions

Memory accesses are primarily controlled through the use of domains, and a secondary control is the access permission set in the page tables. As mentioned before, the first level descriptors could be assigned to one of the 16 domains. When a domain has been assigned to a particular address section, it must obey the domain access rights assigned to that domain. Domain access permissions can be configured through the CP15:c3 register and each of the 16 available domains can have the following bit configurations.

- **Manager (11):** Access to this domain is always allowed
- **Client (01):** Access controlled by permission values set in the page table entry
- **No Access (00):** Access to this domain is always denied

If the configuration is set to Client, it will look at the access permission of the corresponding page table. The table below shows how the MMU interprets the two bits in the AP bit field of the page table.

Table 2-1: Page table AP Configuration

AP bit	User mode	Privileged mode
00	No access	No access
01	No access	Read and write
10	Read only	Read and write
11	Read and write	Read and write

With the help of the domain access control and page-level protection, we can isolate different memory regions in the system to achieve the wanted security configuration.

### 2.1.2.9 Summary

This concludes the ARM architecture background information where we described the fundamental mechanisms necessary to apply virtualization to a system. In the next section, we will go into the detailed design on how the SICS hypervisor is designed and implemented.

## 2.1.3 SICS Hypervisor

The hypervisor software was designed for the ARM architecture, with the main goal to improve the security of an embedded system through the isolation properties that it can provide. This is achieved by only letting the hypervisor execute in privileged mode, where it can control all the systems hardware resources and control the access policy and security of the running application and operating systems. In our case, the SICS hypervisor uses paravirtualization to virtualize the guest system to run on top of the hypervisor in a lower privilege mode (user mode).

By having several virtual guest modes, each with its own execution context and memory access configuration, the hypervisor can enforce the memory isolation between the operating system, its applications and most importantly, the security critical applications.

### 2.1.3.1 Guest Modes

The hypervisor supports arbitrary number of "virtual" guest modes. As guest modes have their own memory configuration and execution context, the hypervisor always controls which current guest mode is executing. There are currently four virtual guest modes defined in the hypervisor:

- **Kernel mode:** Execution context for guest kernel
- **Task:** Execution context for user applications
- **Trusted:** Execution context for security critical applications
- **Interrupted mode:** Execution context for interrupts

These virtual guest modes are necessary in the ARM architecture, because we only have two security rings, privileged and unprivileged. The hypervisor has to reside in the privileged ring while all other software such as, the operating system, task applications and security critical application have to reside in the unprivileged ring. Therefore to keep the separation between the software located in the unprivileged ring, we need these virtual guest modes.

Depending on which the current guest mode is, the memory access to the different domains can be set up differently to suit the security needs of the system. The hypervisor then makes sure that the correct corresponding virtual guest mode is running, depending on whether kernel, task or trusted code is executing. Whenever an interrupt is generated, the hypervisor will change the guest mode to interrupt mode. In the next section we will go through how memory isolation is achieved.

### 2.1.3.2 Memory Protection

With the help of the linker script file, we can decide where the hypervisor, kernel, task and trusted code are placed in the memory. Through the domain access permission and the page table access permission we can setup access rights to different parts of the system by separating the memory addresses into several domains as shown below.



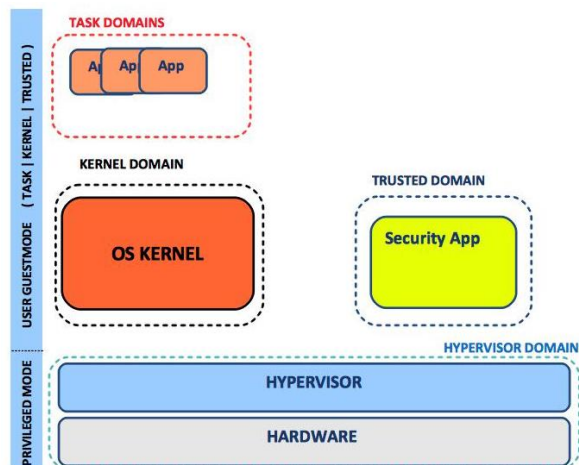


Figure 2-3: MMU Domains

### 2.1.3.2.1 Hypervisor Protection

The hypervisor, boot code, exception handlers and all device peripherals are located in the hypervisor domain. This domain is only accessible in privileged mode, which the system boots up in. At system boot, the hypervisor sets up the hardware and configures the MMU according to the wanted security configurations. Before the hypervisor starts up the guest OS, it makes sure to switch the processor state to user mode and the current virtual guest mode to kernel mode, and continue the execution of the OS kernel application. Transition back to privileged mode only occurs on hypercalls or hardware exceptions, ensuring that no one except the hypervisor can tamper with the memory configurations of the MMU.

### 2.1.3.2.2 OS Kernel Protection

The kernel code and data are located in the kernel domain. When the OS kernel context is running, the hypervisor makes sure that the virtual guest mode kernel is active. The domain access configuration for virtual guest mode kernel is client access to the kernel and application domain. The trusted domain is set to no access. The figure below shows the general memory domain access configuration for the kernel mode.

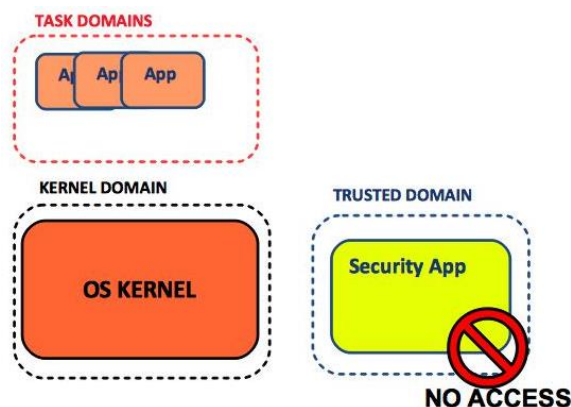


Figure 2-4: Kernel mode domain access

### 2.1.3.2.3 Task Protection

The user application code and data are located in the task domain. When the application context is running, the hypervisor makes sure that the virtual guest mode task is active. The domain access configuration for virtual guest mode task is client access to the task domain. Kernel domain and trusted

domain are both set to no access. This isolates the task from the rest of the system. The figure below shows the memory domain access configuration for the task mode.

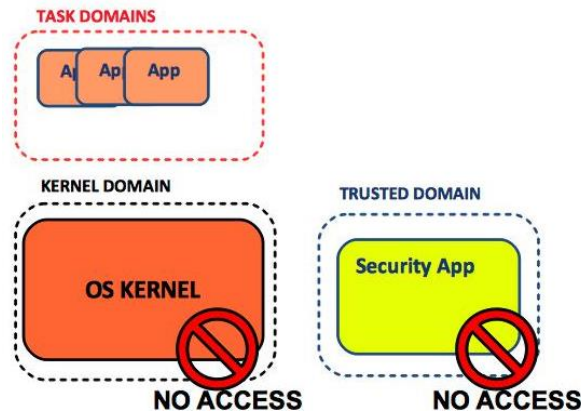


Figure 2-5: Task mode domain access

#### 2.1.3.2.4 Trusted Protection

Lastly, a domain is reserved for our security critical applications. This domain will be completely isolated from all other domains in order to protect all data that reside in it from illegal accesses. When a security application that resides in the trusted domain is running, the hypervisor makes sure that the virtual guest mode trusted is active. The domain access configuration for this mode is client access to trusted domain and no access to all other domains.

To use these secure services, a secure well-defined interface is provided that can be called through a remote procedure call (RPC). This will be described in the next section. A typical scenario is a commodity operating system and an isolated trusted domain offering secure services to untrusted applications.

The figure below shows the memory access configuration for the trusted mode.

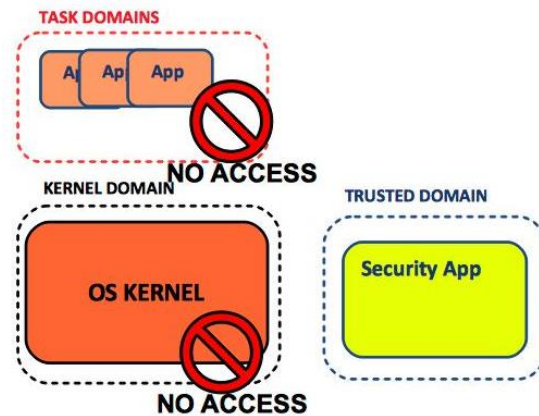


Figure 2-6: Trusted mode domain access

#### 2.1.3.3 Hypervisor Configuration

For each system running on the hypervisor, there is a configurable file that contains the setup of the running system. It contains information regarding which address space and domain each region reside in and capability list of each virtual guest mode, what it is allowed to do and not. This configuration file is parsed by the hypervisor at start-up to setup the hypervisor system. There is one configuration file for each platform and OS port.

### 2.1.3.4 Hypercall Interface

To provide a safe access to privileged functionality, the hypervisor has a hypercall interface available for the guest OS to use. Because the OS kernel now runs in unprivileged mode, all the previous privileged instructions that the kernel could do itself, now has to be delegated to the hypervisor through hypercalls. These are invoked through the SWI instruction. A list of the offered hypercalls can be seen in the table below. Each hypercall is verified by the hypervisor to confirm that it is an allowed operation. Important checks consists of where the call origin from, which virtual guest mode, address space and memory access configurations.

**Table 2-2: Hypercall interface**

Hypercall	Description
BOOT_INFO	Provides the hypervisor with OS specific information to setup the system
SET_CPU_CR	Sets the CPU control register
GET_CPU_CR	Gets the CPU control register
IRQ_SAVE	Saves the state of the IRQ
IRQ_RESTORE	Restores the state of the IRQ
INTERRUPT_SET	Sets the state of the IRQ and FIQ in the CPSR
END_INTERRUPT	Gives back execution to the interrupted context (called after hardware interrupts)
IRQ_DISABLE	Disables IRQ
IRQ_ENABLE	Enables IRQ
FLUSH_ALL	Flushes all caches
TLB_INVALIDATE	Invalidates translation look aside buffer
CACHE_INVALIDATE	Caches invalidate operations on virtual address
SET_PAGE	Page table operations
CREATE_SECTION	Creates a section page table and inserts it in the master page table
RESTORE_USER_REGS	Used to switch execution context
SWITCH_MM	Sets another page table for user process
RPC	Used to communicate between different guest modes. (Used to change virtual guest mode and execute security critical applications).
END_RPC	Gives back execution to the RPC caller

### 2.1.3.5 Hardware Exceptions

Whenever a hardware exception or an interrupt occurs, the processor suspends the ongoing execution context, switches to privileged mode and jumps into the corresponding exception handler. These have all been redirected to the respective hypervisor handler routine.

- **Undefined:** When the hypervisor encounters an undefined exception, it means that the guest OS tried to execute an instruction that the ARM processor could not understand. The hypervisor can either decide to halt the execution or jump over it and continue execution.
- **Software interrupt:** This is the hypercall handler. When the guest OS executes a software interrupt, the handler decodes the requested hypercall and executes the requested operation.

This can also be a user application requesting the OS kernel to perform a system call, in which case the hypervisor redirects execution to the OS kernel.

- **Prefetch abort:** When the guest OS attempts to fetch an instruction from an address without correct access permissions, the hypervisor first check the memory address if its an legal access and redirects the execution to the OS kernel own prefetch abort handler (if available). If it's an illegal access, the hypervisor can halt execution or jump over it.
- **Data abort:** When the guest OS attempts to access data memory without correct access permissions, the hypervisor check the memory address if it's an legal access and redirects it to the OS kernels own data abort handler. If it's an illegal access, the hypervisor can halt execution or jump over it.
- **Interrupt:** Hypervisor interrupt handler saves execution context of the interrupted task, which includes the CPU registers, state and guest mode. It then disables interrupts for the guest OS and redirects to the OS kernels interrupt handler. When the guest OS has finished handling the interrupt, it gives back execution to the hypervisor which restores the execution context of the last interrupted task.

Whenever the hypervisor gets execution, it saves the interrupted virtual guest mode and its execution context in order to know where to resume execution when it is finished with its operations.

### 2.1.3.6 Hypervisor Development Progress

The hypervisor was first developed for the ARMv5 architecture, specifically the ARM926EJ-S CPU supporting the FreeRTOS kernel as a paravirtualized guest. All hardware and peripherals were initially simulated with the simulation software Open Virtual Platforms (OVP) [12]. A generic OVP barebone platform was used.

As the first hypervisor prototype was deemed successful in terms that security was significantly increased because of the added isolation properties, efforts were made to add more platform support, hypervisor functionality and use real hardware. The following are the progress list of the hypervisor.

- Functional hypervisor on ARMv5 926EJ-S CPU with paravirtualized freeRTOS port running on simulation software (OVP)
- Added security critical application running isolated from the rest of the system on top of the hypervisor to test real use case.
- Added hypervisor DMA virtualization to prevent illegal hardware access.
- Added support for ARMv7 Cortex A8 on hypervisor (OVP)
- Modified hypervisor to be more platform and architecture independent through a hardware abstraction layer (HAL). Increased portability.
- Added real hardware support for the hypervisor (BeagleBoard and BeagleBone). Working in real hardware.
- Paravirtualizing Linux kernel and adding hypervisor functionality to support Linux (Current)

### 2.1.4 FreeRTOS Port

FreeRTOS is a free open source real time operating system that we have ported to work on the SICS hypervisor. It is very simple and small, consisting of approximately 4000 lines of code, with no file systems, complex memory management or device drivers.

Thus paravirtualizing FreeRTOS is a relatively simple task, compared to for example the Linux kernel. We will show the main modifications required for it to work on the hypervisor. The figure below shows the basic structure of the FreeRTOS system.

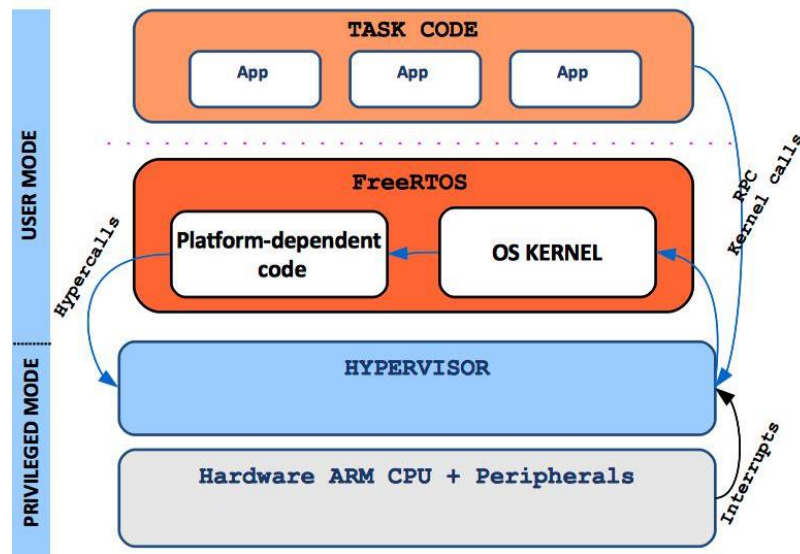


Figure 2-7: FreeRTOS hypervisor system

#### 2.1.4.1 FreeRTOS Kernel

The core FreeRTOS kernel has remained almost completely unchanged except from some minor modifications on how the task applications are allocated. Previously, the kernel allocated memory for all tasks from the same heap. We added extra functionality to allocate task memory from a pool of separated heaps which also gives you the possibility to create isolation between the different application tasks. Except from this change, the core kernel has stayed unmodified.

#### 2.1.4.2 Platform Dependent Code

The platform dependent code is responsible for carrying out critical low-level operations, which requires privileged instructions. These have been paravirtualized replacing all the privileged instructions with hypercalls.

#### 2.1.4.3 Memory Management

Originally, FreeRTOS does not have any memory protection, as it does not use the MMU to setup the system. All memory accesses use physical addresses and are unprotected. This means that security is non-existent for the system as user applications have full access to all hardware resources and the kernel. Because there were no memory management for the FreeRTOS we had full freedom to decide the location of all software regions, access control and how the memory addresses will be mapped.

What we have done is through the linker script, defined distinct addresses for the different software regions (kernel address space and user task address space), and through the MMU, domain and page table settings, setup a configuration that effectively enforces an access policy that provides us with memory isolation between our hypervisor, OS kernel and user applications. This is used in our first prototype for the nSHIELD 3.3 deliverable and more details on the configuration can be seen in D3.3.

#### 2.1.4.4 Hypercalls / System Calls

As the FreeRTOS originally did not have any kernel and user space separation, user applications could directly use the kernel API to call kernel functions. This is generally not the case in more complex operating systems where applications have to use system calls through the SWI instruction to perform kernel operations.

With the hypervisor in place creating separation between kernel and user address space, it can no longer access the kernel API. To solve this, the kernel API functions have been wrapped around RPC hypercalls

that changes the virtual guest mode, this in order to get access to the kernel address space. This provides a secure interface to use the kernel API without compromising the security of the kernel. When the kernel API call is finished, an end RPC hypercall is issued to change the current guest mode back to task mode, disabling the kernel domain and yielding back to the calling task.

#### 2.1.4.5 Interrupt Handling

The FreeRTOS scheduler relies on the hardware timer to schedule tasks. In order for FreeRTOS to work as usual, the Hypervisor redirects all timer interrupts to the FreeRTOS timer tick handler, which schedules the next task.

When FreeRTOS is scheduling a new task, it saves the current tasks execution context in its data structures and uses a hypercall to load the next running task context.

#### 2.1.4.6 Summary

In our experience, paravirtualizing the very small OS FreeRTOS was relatively easy to accomplish. We went from a system with non-existent security to a hypervisor protected secure environment.

### 2.1.5 Linux Port

Work is currently in progress of porting the Linux kernel version 2.6.34.3 to our hypervisor using paravirtualization. The main reason behind choosing this specific kernel release is that the simulation software OVP already has a working Linux kernel running on an ARM Cortex A8 Integrator CP platform. To simplify things, the Linux kernel 2.6.34.3 has been modified and compiled with only UART support as a start. The kernel starts the bash shell command that communicates with a serial console through the UART. This will be the start base of our porting efforts. We will issue the biggest challenges with the port.

- Memory management
- Interrupt service routines
- Processes
- Hypercalls & System Calls
- Platform support

#### 2.1.5.1 Memory Management

One requirement for Linux to work is that it requires a one to one mapping of the physical memory to the kernels virtual address space. This is given to the Linux kernel by the hypervisor during the boot/initialization phase. The hypervisor provides the page table structure for the OS and keep track of which physical memory addresses that belong to the OS. Any attempts to access memory outside of the OS will generate an access fault. Currently, the configuration is static in the first iteration of the port and more advanced features can be added when we have a more stable version. As the current goal is to successfully run a single guest, we don't need any advanced OS like memory management in the hypervisor.

As the Linux kernel no longer has permission to communicate with the MMU, it has to go through the hypervisor to update these parts of the CPU. The hypervisor will also be responsible for updating the Linux page table whenever it needs more memory, or when there is free memory that it no longer needs. Another instance is when a certain application wants access to something in memory that is not yet mapped to physical memory; it will generate a page fault. The hypervisor need to let the Linux kernel know that it must look up the physical address to map it into the specific application. As the true hardware page tables are kept inside the hypervisor the OS have a shadow page table to be able to keep track of the page attributes in order to properly map and unmap pages for the user processes.

Modifications to the memory management of the Linux kernel also includes giving the SICS hypervisor responsibility for all CPU operations such as flushing/invalidating the TLB and cache.

### 2.1.5.1.1 Page Tables in ARM Linux

Hardware wise, ARM has a two level page table structure, where the first level has 4096 entries, and the second level has 256 entries<sup>4</sup>. Each entry is one 32-bit word. The problem here is that the page tables do not contain any "accessed" or "dirty" bits which specify if the address have been accessed or written. The Linux kernel uses this information for its memory management operations such as resolving page faults, freeing memory and swapping memory. Thus, Linux has to keep its own copy of the page tables that contains the missing information. Another detail is that Linux has a three level page table structure<sup>5</sup> which in the ARM port is wrapped to fit in a two level page table structure using only the page global directory (PGD) and page table entry (PTE).

Linux therefore is tweaked to having 2048 entries in the first level, each of which is 8 bytes (i.e. two hardware pointers to the second level). The second level contains two hardware PTE tables (of 256 entries each) arranged contiguously, followed by Linux version which contain the state information that Linux needs. Therefore, the PTE level ends up with 512 entries [13].

### 2.1.5.1.2 Linux Kernel Mappings

The Linux kernel has a very specific virtual memory layout that we have to follow, in order to not break kernel functionality. Mappings, which collide with the above areas, may result in a non-bootable kernel, or may cause the kernel to eventually panic at run time.

Looking at the table below [14], we can see that the virtual address space of a user process in native Linux kernel range from 0x1000 to PAGE\_OFFSET<sup>6</sup> - 1. This address space can be accessed in either user or privileged mode. The address space from PAGE\_OFFSET to 0xFFFF FFFF belongs to the kernel and can only be accessed in privileged mode. The hypervisor virtual guest modes will mimic these privilege states in order to enforce these rules.

The hypervisor software also needs a mapping and can be put between VMALLOC\_END - 0xFEFFFFFFF, which is free for platform use. We have decided that a suitable location is 1MB from 0xF000 0000 is mapped for the hypervisor.

**Table 2-3: Kernel Memory Layout on ARM Linux**

Start	End	Use
0xFFFF 8000	0xFFFF FFFF	Copy_user_page / Clear_user_page use
0xFFFF 4000	0xFFFF FFFF	Cache aliasing on ARMv6 and later CPUS
0xFFFF 1000	0xFFFF 7FFF	Reserved. Platforms must not use this address range
0xFFFF 0000	0xFFFF 0FFF	CPU vector page. The CPU vectors are mapped here if the CPU supports vector relocation (control register V bit)
0xFFFE 0000	0xFFFF 0FFF	XScale cache flushes area. This is used in proc-xscale.S to flush the whole data cache. (XScale does not have TCM.)
0xFFFE 8000	0xFFFE FFFF	DTCM mapping area for platforms with DTCM mounted inside the CPU

<sup>4</sup> Linux ARM kernel only uses small pages of 4KB for second level descriptors

<sup>5</sup> This comes from the x86 architecture

<sup>6</sup> Normally set to 0xC000 0000 which gives a user/kernel split of 3GB/1GB

0xFFFF E000	0xFFFE 7FFF	ITCM mapping area for platforms with ITCM mounted inside the CPU
0xFFF0 0000	0xFFFD FFFF	Fixmap mapping region. Addresses provided by <code>fix_to_virt()</code> will be located here
0xFFC0 0000	0xFFEF FFFF	DMA memory mapping region. Memory returned by the <code>dma_alloc_xxx</code> functions will be dynamically mapped here.
0xFF00 0000	0xFFBF FFFF	Reserved for future expansion of DMA mapping region
VMALLOC_END	0xFEFF FFFF	Free for platform use, recommended. VMALLOC_END must be aligned to a 2MB boundary.
VMALLOC_START	VMALLOC_END -1	<code>vmalloc()</code> / <code>ioremap()</code> space. Memory returned by <code>vmalloc/ioremap</code> will be dynamically placed in this region. VMALLOC_START may be based upon the value of the <code>high_memory</code> variable
PAGE_OFFSET	<code>high_memory -1</code>	Kernel direct-mapped RAM region. This maps the platform RAM, and typically maps all platforms RAM in a 1:1 relationship.
PKMAP_BASE	PAGE_OFFSET -1	Permanent kernel mappings. One way of mapping HIGHMEM pages into kernel space.
MODULES_VADDR	MODULES_END -1	Kernel module space Kernel modules inserted via <code>insmod</code> are placed here using dynamic mappings
0x0000 1000	TASK_SIZE - 1	User space mappings Per-thread mappings are placed here via the <code>mmap()</code> system call
0x0	0x0000 0FFF	CPU vector page / null pointer trap CPUs, which do not support vector remapping, place their vector page here. NULL pointer dereferences by both the kernel and user space is also caught via this mapping.

### 2.1.5.1.3 System Memory Layout

The current memory layout looks the following.

Physical address 0-1MB is reserved for the hypervisor while address 1-15MB can be reserved for the trusted applications that provide security services to the Linux kernel and shared memory. The reason we have assigned so much to the trusted applications and shared space is because the Linux kernel start address is required to be a multiple of 16MB. We used this solution, as it was relatively easy to configure the Linux start address to move 16MB forward while keeping the hypervisor at its original address at 0. This might change in the future to better utilize RAM.

The Linux kernel then has from 16MB - end of RAM to claim for its own use. Approximately the first two megabytes of physical memory are reserved for hardware/boot information, Linux OS text, data and paging data structures. Below is a rough overview on how the physical- virtual mappings look like.



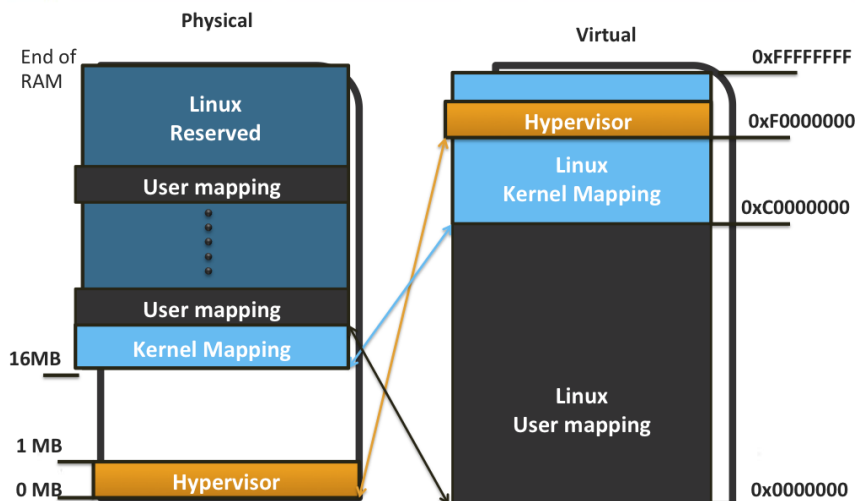


Figure 2-8: System memory layout

2.1.5.1.4 Memory Configuration

The table below shows how the different regions have been setup in the page tables.

Table 2-4: Page table access permissions configuration

Region	Domain	AP (User)	AP (Privileged)
Hypervisor	0	No Access	Read/Write
Kernel	1	Read/Write	Read/Write
User	2	Read/Write	Read/Write
IO	3	Read/Write	Read/Write
Trusted	4	Read/Write	Read/Write

The table below shows the domain access configuration of the system. The different virtual guest modes set the access permissions to the different domains. 00 specifies no access, 01 checks the access permissions in the page table according to the table above. As usual, only virtual guest mode trusted has access to the trusted domain. Virtual guest mode kernel has access to its own kernel domain, user and IO, while virtual guest mode User has access to its own user domain and IO.

Table 2-5: Domain access configuration

Mem Region	TRUSTED	IO	USER	KERNEL	HYPERVISOR
Domain	4	3	2	1	0
Virtual Guest Mode					
GM_TRUSTED	01	00	00	00	01
GM_KERNEL	00	01	01	01	01
GM_USER	00	01	01	00	01

2.1.5.2 Interrupt Handling

The hypervisor have to deal with the interrupts, as it is the only software that has privilege to handle it and pass the hardware information to the Linux kernel top-half interrupt handler where the interrupt is

processed. It runs with the highest priority and can't be preempted or sleep which makes it important that the interrupt is processed as fast as possible. Another thread executes all bottom halves once the pending top halves have been completed.

In the Linux kernel, there is interrupt handlers for both user and privileged modes, and the handlers also switch to supervisor mode (SVC) after entering an interrupt. Before switching, it copies the IRQ registers to the SVC registers. This makes it possible to handle another interrupt while the other is getting served in SVC mode. The hypervisor interrupt handling has been rewritten to support the same mechanism. However, instead of switching to SVC mode it switches to user mode when jumping into the Linux kernel, making it able to catch another interrupt.

The Linux kernel also has its own mechanism to register an interrupt with the system and provides means to acknowledge and clear interrupts. These are provided through hypercalls. Critical low level interrupts like the timer controller is handled directly in the hypervisors interrupt vector. To enable a fast and responsive system, it is important the timer interrupt is redirected as fast as possible to the Kernel tick handler in where it updates time and execute the next scheduled task/process.

### 2.1.5.3 Processes

Each process has its own Page Global Directory (First level descriptor in ARM) and its own set of page tables. Whenever a process switch occurs, page tables for user space are switched as well in order to install a new address space. This will be handled in the hypervisor.

The kernel allocates the virtual addresses to a process called memory regions. Typical situations in which a process gets a new memory region is creating a new process (fork), loading a new program (execve), memory mapping (mmap), creating shared memory (shmat), expanding its heap (malloc) etc. The linear address space of a process is divided into two parts. We have user processes that are allowed to run in the virtual address space between 0x0 to 0xBFFF FFFF and kernel processes that can run from virtual address 0xC000 0000 to 0xFFFF FFFF, each running in their corresponding virtual guest mode (virtual user and virtual kernel mode). One important exception is the first MB in 0xF0000000, which is reserved for the SICS hypervisor, this address space can only be run in privileged mode of the processor.

The Linux kernel will remain the pager for the user tasks and any user process page fault will be redirected from the hypervisor to the Linux kernel in order to decide whether it's legal. The Linux pager works in a way called demand paging. This meaning a process only gets additional virtual address space, not physical when it requests more memory from the kernel. When it actually tries to access the new virtual address, a page fault occurs to tell the kernel that the memory is actually needed (i.e. demand paging) and maps the pages of its address space to the Linux user process.

So, during execution at user level, only user space should be accessible. Attempting to read, write, or execute kernel space/hypervisor space shall cause an exception that the hypervisor takes care of. In contrast, during execution in the Linux kernel, both user and Linux kernel space are accessible. Hypervisor will have access to the whole address space. The correct virtual guest mode will be enforced and handled by the hypervisor in order to provide the correct access configurations.

### 2.1.5.4 Hypercalls / System Calls

As stated earlier, when using the para-virtualization method, the Linux kernel needs to be modified to be able to run in user mode. Not being able to execute privileged instructions anymore, the hypervisor have to provide a mechanism to execute these instructions, which is called hypercalls. This is the operating systems counterpart to the system call, which is used in applications.

When the Linux kernel executes a hypercall, it puts data structures, operators and other arguments to the specific hypercall in the ARM registers R0-R1 while R2 is reserved for the hypercall number. The SWI handler then interprets these parameters.

It's also important that user processes can run on the Linux kernel unmodified, maintaining binary compatibility. This is achieved by reserving all the original system call numbers from Linux in the hypervisor. When the user process uses the library to use the system calls, which is done through a SWI

instruction, the hypervisor SWI handler catches this and forwards the call to the Linux kernel with the correct virtual guest mode where it will be processed. When it is finished with the kernel operation, it gives back execution to the hypervisor that in turn restore the context of the user process that used the system call.

There are also some important security checks in these handlers checking if the hypercall originated from the correct address space, such as that user space can never execute hypercalls that is intended for the kernel. Another important thing is that if a pointer is passed to the hypervisor, it needs to make sure that the process executing the call owns the memory address of the pointer. It would be a security hole if it were possible to pass arbitrary memory address and the hypervisor blindly writes to the given parameter. Currently, we check the virtual address of the running process along with the active virtual guest mode. If the virtual address is smaller than 0xC0000000, it belongs to user space, and above 0xC0000000 belongs to kernel, except the first MB from 0xF0000000, which is the hypervisors virtual address space.

### 2.1.5.5 Linux Kernel Integrator Platform

Normally the platform setup is performed in the start of the Linux kernel; it is now being setup by the hypervisor initialization and is not executed in the Linux kernel. Only the relevant data structures and device operations are kept to maintain normal functionality. The platform IO has been setup to use the following mappings.

**Table 2-6: Integrator CP platform IO**

Virtual	Physical	Device
0xF100 0000	0x1000 0000	Core module registers
0xF110 0000	0x1100 0000	System controller registers
0xF120 0000	0x1200 0000	EBI registers
0xF130 0000	0x1300 0000	Counter/Timer
0xF140 0000	0x1400 0000	Interrupt Controller
0xF160 0000	0x1600 0000	UART 0
0xF170 0000	0x1700 0000	UART 1
0xF1A0 0000	0x1A00 0000	Debug LEDs
0xF1B0 0000	0x1B00 0000	GPIO

The IO has been setup by the hypervisor and complies with the virtual addresses of Linux.

### 2.1.5.6 Summary

The Linux port is still ongoing work and in this phase of development, it's still too early to present a prototype. However, foundation of the port is in place and we have gotten as far as the kernel has successfully passed all boot and architecture initialization, exception handlers are connected through the hypervisor, major part of the hypercalls and system are paravirtualized and the scheduler has started the first user process. As the memory management is quite complex in the Linux kernel, we have to go through extensive debugging and tests in order to make sure that it works as intended, especially the user processes running in the system.

When we have managed to get the bash shell command up and functioning, the next step is then to put the ported OS through extensive system tests and benchmarks to be sure that it's stable and dependable.

## 2.2 Secure ES firmware

### 2.2.1 Description

The prototype hardware is based on the jointly selected BeagleBone platform which includes an AM3359 processor (ARM Cortex-A8). The BeagleBone development kit is shipped with software including a boot loader and Linux. The actual boot loader is UBOOT and it lacks some properties for required by nSHIELD. There is no integrity control of the images to be loaded; the footprint of the final image exceeds the internal static ram inside the ARM processor (which has resulted in a two stage boot procedure). Finally the code base is licensed under the restrictive GPL license. Therefore a new design has been selected in favour of modification of UBOOT. The high level requirement from nSHIELD to apply SPD Metrics on the firmware must be addressed in the design. Since the final design goal is to provide a ROM code inside the ARM processor the dependency of other nSHIELD components is not applicable. Privacy depends on integrity. General security metrics could be mapped to properties of cryptography such as selected algorithms and key length, however security also depend on the quality of the implementation, therefore the design is made for review. The design enlightens the execution path from power-on until control is passed to the loaded and verified image. The flexibility of Uboot is provided by parsing custom boot scripts; in the new design one static boot flow applies restrictions, but enables improved security instead. Flexibility is however provided by the layout of the image, since it contains different objects such as executable, ram disk and parameter blocks. Each object in the image is cryptographically sealed. The parameter block enables allocation of loaded objects in memory as well as passing parameters to the object prior to execution.

The first software that executes after a power-up or a reset is secloader. The name indicates that this is prototype and not implemented inside the ARM processor. The functionality of secloader can be visualized with the flow chart depicted in Figure 2-9.

When the system is started, secloader first initializes hardware and starts the watchdog. After this, a normal power-on-self-test should be performed.

When the power-on-self-test has executed, secloader checks if a secondary boot media (USB stick) is present. Before booting from the USB stick, secloader checks how many times it has previously tried to boot from the USB stick. For each attempt, the `dirty_external` parameter is incremented by one. If this parameter reaches `max_external`, secloader no longer tries to boot from the USB stick. Instead, it attempts to boot from the primary media.

When booting from USB, secloader searches for an image on the USB stick. To verify the images, secloader uses the public part of the key, which is stored on the board. If secloader fails to authenticate the images, external boot is aborted and secloader tries to boot from internal media. If the images are correctly verified, secloader increments the `dirty_external` parameter and extracts them to internal RAM. After this is done, the extracted and verified image is started.

If no USB stick is detected, secloader attempts to boot from the internal media. As is the case when booting from USB, secloader checks how many times it has tried to boot from internal media by verifying and thereafter incrementing the `dirty_internal` parameter. If the parameter reaches `max_internal`, secloader no longer tries to boot from compact flash.

When booting from compact flash, the image is located on the first partition of the media. If secloader fails to authenticate the image, internal boot is aborted and secloader performs a hard reset of the board. If the images are authenticated, they are extracted to internal RAM and started in the same way as for USB.

If both primary and secondary media are corrupt i.e. both `dirty_internal` and `dirty_external` have reached their respective max-values, `dirty_external` is set to zero before the board is reset.

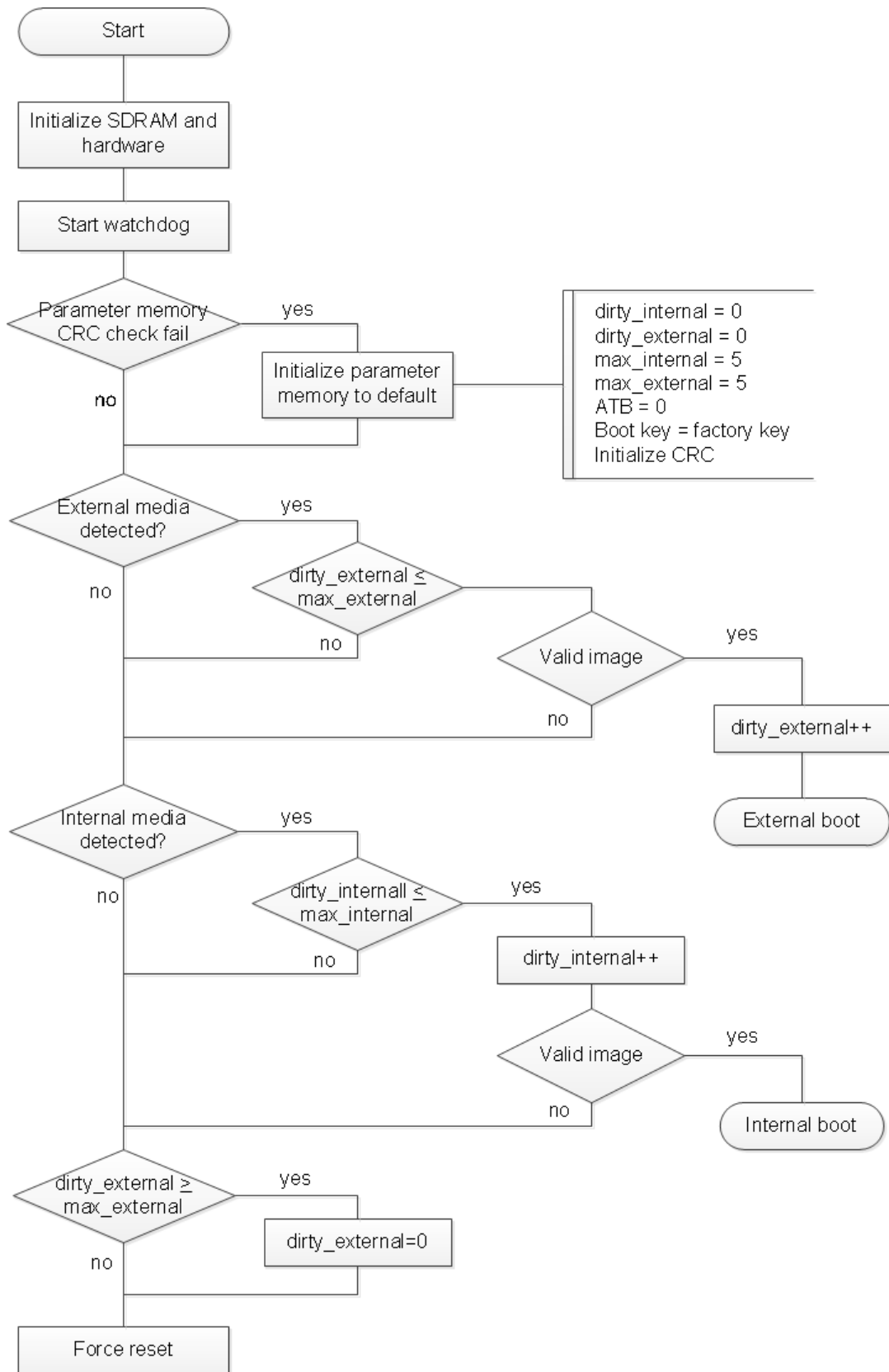


Figure 2-9: Secloader functionality

## 2.2.2 Architecture Modules

The secure firmware is based on a fixed boot flow that enables addition of cryptographic functions for verifying issuer of images to be loaded. The implementation of the prototype is written mainly in portable C with a few exceptions:

- Low level hardware access
- CPU mode and cache
- Stack management

The resulting firmware image is monolithic but is based on several modules. The execution path involves two modules:

- bios\_low (memory and hardware setup)
- bios\_high (static boot flow logic)

The first instruction in the bios\_low module is placed on address 0x402F0400 in the processors physical address space. The last instruction is a call to bios\_high. In between a full C environment is established so that code generated by a standard C-compiler could be used for all functions in bios\_high. To allow flexible memory configuration the new design will adopt Serial Presence Detection. Normally a memory module has a small i2c module containing all sizing and timing parameters needed for memory controller configuration. The prototype will use the i2c memory located on the motherboard to simulate a memory module. The static specific memory configuration will be replaced with a standard SPD detection similar to X86 based systems.

The remaining modules are:

- bios\_timer: timing function library and hardware setup
- i2c\_driver: management of devices on i2c bus
- bios\_mmc: media controller (SD card)
- bios\_output: output on serial port
- plf\_bios\_arm: minimal portable C-library
- provision: cryptographic library
- bios\_vfat: file system library
- bios\_usb: usb stack for storage type device

## 2.2.3 Architecture Interfaces

### 2.2.3.1 Passing SPD Metrics

The firmware is responsible of configuring SDRAM and then report the size of memory to the object that will be loaded and executed. For ARM processors a data structure in memory holds this information. By adding a new type to the ATAG-list, SPD metrics could be passed from the firmware to next layer of software (Linux / hypervisor).

### 2.2.3.2 Load Image Format

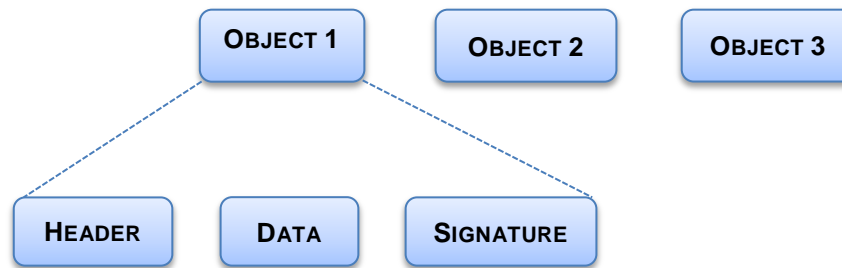


Figure 2-10: Load image format

```

struct linux_kernel
{
    int load_address;
    int start_address;
    char cmdline[1024];
};

struct hypervisor
{
    int load_address;
    int start_address;
    char cmdline[1024];
};

typedef union
{
    struct linux_kernel;
    struct hypervisor;
} item;

typedef struct object
{
    int type;
    item data;
    int object_start;
    int object_len;
    int object_signature_type;
    int object_signature_start;
    int object_signature_len;
};
  
```

### 2.2.4 Boot Media Layout

By dividing the boot media into partitions and optionally keep the boot partition read-only, the solution becomes more robust.

The production file system is located on the second partition.

### 2.2.5 Parameters Stored on the Motherboard

The parameter memory is located in a non-volatile I2C memory located on the board. It is available from secloader, and must be accessible from the next layer of software. The following parameters are stored in the parameter memory:

**Table 2-7: Parameter memory contents**

Name	Description
Serial	The serial number written during production of the unit.
Boot key	The public part of the key used to verify images to load.
ATB	Attempt-to-boot flag.
dirty_internal	A parameter indicating how many times secloader has tried to boot from compact flash.
max_internal	The maximum number of internal media boot attempt permitted.
dirty_external	A parameter indicating how many times secloader has tried to boot from USB.
max_external	The maximum number of USB boot attempt permitted.

The parameter memory has a CRC that guarantees that the data is not corrupt. If the CRC is not correct, secloader initiates the parameter memory to default values.

### 2.2.6 Metrics

SPD Values:

- The firmware resides inside the silicon of the ARM processor (last phase in design document) and dependencies to other nSHIELD components are therefore not applicable. Code quality is subject to analyse, other metrics include code size, complexity, test coverage etc.

Dependability:

- Not applicable

Privacy:

- Not applicable

Security:

- Code Quality 1-255
- Strength of cryptographic algorithms 1-255
- Length of cryptographic keys 1-255

SPD Metrics will be passed as ATAG list element.

## 2.3 Smart Card Security Services in nSHIELD

### 2.3.1 Building Secure Communications

In order to build trust among different type of nodes on the nSHIELD architecture we can exploit the benefits of smart cards and the cryptographic schemes they implement. Considering, the nSHIELD architecture where decentralized components are interacting not only with each other but also with centralized ones, depending on the type of the device and the employed scenario; there is a need for integrating security and interoperability. In this context, we rely on [15] for building secure communication channels among different devices. We should mention that smart cards currently are used in various applications, where proof-tamper devices are need for the provision of security services.



In the proposed scheme in order to issue a smart card the related component (e.g. micro node) should create a request for issuing a smart card. This request will include the serial number of the component and will be forwarded to the central authority. If needed, depending on the type of service, the central authority will check the register status of the requested component and afterwards will generate a new smart card. In the new smart card will be installed the following information:

- Node's serial number.
- Node's secret key.
- Node's id.
- Node's auth key.

The generation of secret keys will be based on the following types:

Encryption-Key = AES-256 (Central Mother Key XOR Node's Serial Number)
Auth-Key = AES-256 (Central Mother Key XOR Node's ID)

We should note that in this scheme we assume that the central authority has also a TPM for generating the secret keys in a secure way for the issued tokens, while all the smart cards are issued by a (trusted) central authority. The generated keys will be unique since they are related with node's serial number, which is unique.

The node, as a smart card is issued can exploit its security feature for providing confidentiality, integrity or/and authenticity services. The provided services depend on the application. For instance, if there is a requirement to provide confidentiality services to the data sent to the central authority the following procedure will be take place:

1. The node will send the data to the smart card.
2. The smart card encrypts the provided data, using the secret-key installed into the smart card during the registration, and forwards them to the node.
3. The node sends to the central authority the encrypted data and its serial number.
4. The central authority generates in the TPM the corresponding secret key using the serial number sent by the node. Note that the key is not "extracted" from the TPM.
5. The TPM decrypts the data and send and acknowledgement to the node.

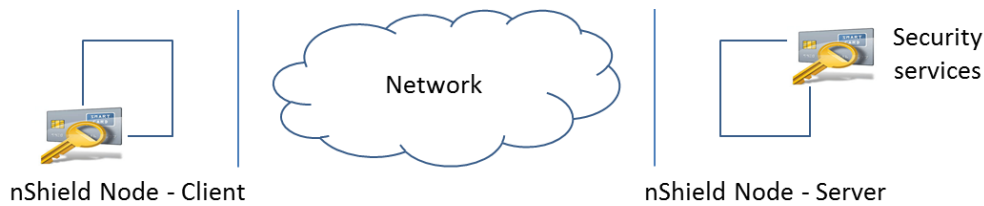
A very similar approach will be followed when an authentication is needed. Particularly:

1. The node will send to the smart card a random number that will be used as the data require validation.
2. The smart card using the Auth-Key and the random data generates the MAC and forwards it to the node.
3. The node sends to the central authority the MAC including the random number and its id.
4. The central authority generates in the TPM the corresponding auth key using the id.
5. The central authority using the auth-key produces a new MAC and compares it with the one received by the node. If those two MACs are matched the central authority sends to the node a successful response otherwise a failure occurs.

These procedures can be combined in order to provide confidentiality and authenticity services simultaneously, depending on the requirements.

## 2.3.2 Implementation

To support the functionality of secure communication described in the above section we are implementing the architecture illustrated in the following figure:



**Figure 2-11: High level components of the proposed implementation**

Particularly, the implemented architecture will be consisted of the following modules:

- Security services
- Server, and
- Client

### 2.3.2.1 Security services

The security services in this architecture will be provided using GPK 16000 smart cards. Currently, we are under the development of the required functionality. The development is taking place on Linux based system relying on the interface of open smartcard [71].

### 2.3.2.2 Server & Client

The server and the client modules are under development.

## 2.4 Power Management & Supply Protection Technology

### 2.4.1 Description

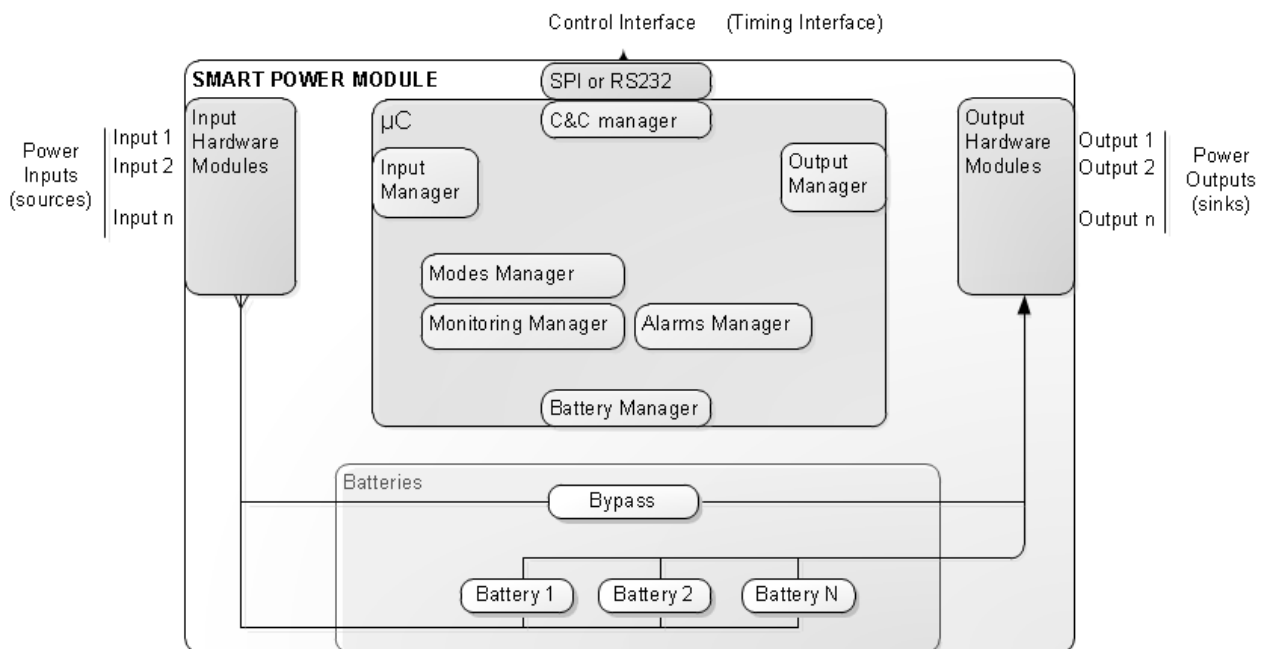
Power management and supply protection is fundamental feature to ensure the reliability of any electronic device. The technology under development uses commercial available power management semiconductors to integrate their features into nSHIELD networks, [16], [17], [18] and [19].

#### 2.4.1.1 Technology Architecture

The first step has been the definition of reference architecture for the power management & supply protection unit as show in Figure 2-12. This power unit is referenced from here on as the Smart Power Unit (SPU). The following sub-modules can be identified in the SPU:

- Power Inputs: Include all the power sources AC/DC and they can be of different types: power line / harvesting / battery ...
- Power Outputs: Include all the power sinks and will be mainly DC in the 3.3V to 12V range.
- Input Hardware Modules: Embed the needed semiconductors to adapt the input power sources to the battery module.
- Microcontroller ( $\mu\text{C}$ ): Includes an ARM low power microcontroller that manages the SPU and implements its needed functions and external/internal interfaces. The following SW/HW have been identified:
  - Input Manager: Manages the input hardware modules.
  - Output Manager: Manages the output hardware modules.

- Command & Control Manager: Implement the command and control interfaces that allow the integration of the SPU with other nSHIELD modules in order to compose the nSHIELD node. Currently standard serial HW interfaces have been proposed (SPI or RS232).
- Modes Manager: Different power modules manager to implement different power states for the SPU (standby, sleep, low power, normal mode...).
- Monitoring Manager: Implements the monitoring of the different HW modules of the SPU. The monitored parameters are accessible externally over the C&C interface.
- Alarms Manager: Implements the alarm configuration and triggering for the monitored parameters.
- Battery Manager: Manages the battery hardware.
- Output Hardware Modules: Embed the needed semiconductors, mainly linear regulators, to provide several output voltage signals. This module support programmable output voltages with different maximum output currents to supply different modules.
- Batteries: Embed all the power storage devices.
  - Bypass: Module that bypass the input power signal to the output power modules for scenarios with no battery units.
  - Battery 1, 2...N: One or more battery devices that provides energy storage.



**Figure 2-12: Smart power unit**

The parameters/attributes that have been identified to define the SPU features and current state are:

- C&C manager:
  - firmware version,
  - SPU name,
  - serial configuration,
  - authentication parameters,
- Input: number of inputs. For each input

- 
- enable/disable,
  - type (Harvesting, Power Line),
  - frequency (DC, AC (frequency))
  - Output: number outputs. For each output
    - enable/disable
    - DC voltage (range)
  - Battery Manager: For each battery
    - enable/disable,
    - capacity,
    - load (%),
    - estimated autonomy,
  - Monitoring:
    - inputs voltages/currents,
    - output voltages/currents,
    - temperature, humidity
  - Alarms:
    - alarm status (related to timestamp if possible):
      - Batteries alarms: Battery Low, Overload, Over Temperature, Input/output over limits
      - Input/outputs alarm: Current threshold overrun

#### 2.4.1.2 Technology Prototype

The previous section describes a general SPU that could not be very energy efficient to implement for specific input/output power signals. In order to develop more energy efficient and simple SPU several more specific devices are planned. The final selection will depend on the requirements and needs of the components that will use this module:

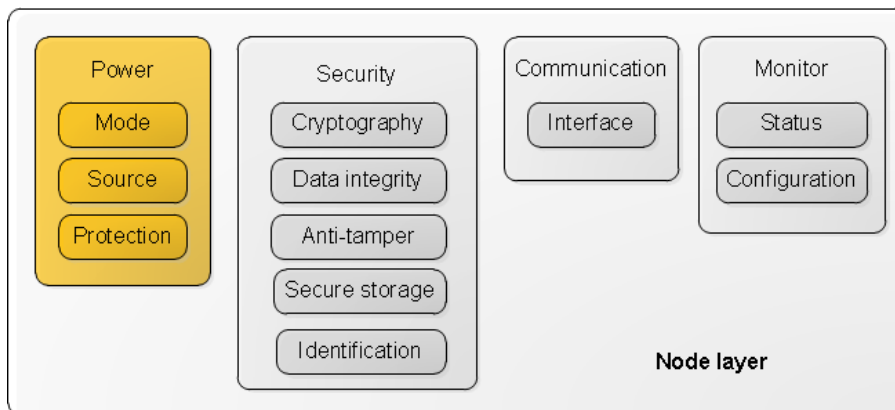
- DC/DC SPU: Power management and supply protection for DC (12V-25V) to DC (3.3V-5.0V)
- AC/DC SPU: Power management and supply protection for AC (220V-50Hz) to DC (12V-25V)
- DC/DC SPU+: Same features of the DC/DC SPU but includes a battery unit.
- AC/DC SPU+: Same features of the AC/DC SPU but includes a battery unit.

All the above SPU include the full implementation of the C&C interface and all the  $\mu$ C internal modules depicted in Figure 2-12.

#### 2.4.2 Architecture Modules and Interfaces

Deliverable D2.4, done in the scope of WP2, defines a set of basic functionalities at node layer in order to support the final composition of capabilities in nSHIELD framework.

In this section, a preliminary design of a power unit has been presented. This unit could be mapped directly as one of the modules, "**power module**", defined in the node layer of nSHIELD device. Figure 2-13 summarized node layer modules described in D2.4.



**Figure 2-13: nSHIELD Reference architecture – Node Layer (power module)**

Taking into account the description of the node layer modules done in the architecture definition, this module should provide some attributes in order to be monitored by other components in nSHIELD architecture.

The components and attributes defined for the power module in the reference architecture are summarized in the following table:

**Table 2-8: Power module at node layer**

		Attributes				
Power module	Mode	Low power				
	Source	Power line	Battery	Energy harvest	AC	DC
	Protection	Fuse				

All of these attributes could be monitored by the monitor module of nSHIELD node, through an interface such as SPI or UART.

Taking account that the definition of the reference architecture proposed at this moment is an intermediate version, the components and attributes can be modified and completed.

Some attributes that could be monitored in order to provide interesting information about the operation of the module would be: inputs voltages/currents, output voltages/currents, temperature, and humidity. These attributes will be finalized taking into account the requirements of the applications scenarios.

### 2.4.3 Metrics

In the same way that D2.4 describes an intermediate version of the reference architecture of nSHIELD system, D2.5 defined the preliminary metrics that nSHIELD system should provide. This preliminary definition of the metrics is split by layer.

The main metrics that can be provided by the module described in this section are related with the *availability*, the *code execution*, the *data freshness*, and the *alternative power supply sources*. Also metrics like *low power mode* would be provided if it is required.

## 2.5 MMC/SD Password Management

For nano node evaluation, in terms of relation between power figures and security, the main reference platform chosen is the "Beagleboard" embedded system (powered by a SoC built around an ARM core).

However, to avoid limiting the exploration to a single case study, we also adopted a virtual platform, based on a customized variant of a software emulator ("qemu"), which is still focused on the ARM architecture. Using a virtual platform is also beneficial because it allows a deep inspection of the HW/SW interaction (by analysing the hardware behaviour even in components which do not expose debug features, as JTAG probing and scan access). Furthermore, within the software emulator, also hardware components not yet developed can be taken into account, and, finally, faulting hardware can be modelled. As reference operating system, the Linux kernel 3.4.4 was selected and ported on the real target system, as well as on the virtual platform.

Mobile nodes must take into account power supply limitations and are also exposed to physical attacks. These two issues are related when data security must be enforced, since a mobile node must be able to safely shut down the elaboration, it must also store data in a persistent memory. Such a storing can trigger a data leak, if not carefully considered, because the storage content could be dumped by an attacker. A possible solution to such an issue is to strongly encrypt all data before storing. However, this approach must pay an overhead both in energy and in computation time that, whenever the power supply is going to be removed, could turn out excessive. A more flexible approach is to recognize that, for some data, cryptography is an overkilling measure. Some information, in fact, must only remain secure for "enough time", after which they become no more relevant. In such a view, the simple password protection offered by standard "Multimedia Cards" (MMC) or "Secure Digital" (SD) memory cards can be secure enough. A node which is shutting down can therefore encrypt only the most sensitive data, and then store encrypted and unencrypted data in a password protected memory card.

Since in the standard Linux kernel, the support for password management of MMC/SD cards is missing, we developed a kernel module to handle this kind of interaction. Such a kernel module implements a simple char driver which provides an ioctl call for receiving commands from user space. Besides the functionality to implement, the other two major objectives were a minimal impact on the kernel code and the portability across different kernel versions.

The low-level interaction with the mmc Linux subsystem requires accessing to some functions that are not exported, and then not available to modules. Rather than patching the kernel sources, which would require a continuous management effort to guarantee the portability, we chose to dynamically search the addresses of the needed functions by exploiting the kallsyms subsystem which, therefore, must be enabled on the target OS (as usually is). Such a search is performed during the device opening (the driver provides the "open" function).

The password management is then performed by the ioctl call, which receives data from user space (the name of the hardware host to use, the password and the action to perform), looks for the required host and then sends commands as lock/unlock and password setting.

The driver was tested on several versions of the Linux kernel, to assess its portability. The only change that can be needed is a slightly modification on the prototype of the ioctl call, since it has been changed when Linux kernel switched from version 2.6.xx to version 3.xx.

## 2.6 User Level Power Management

Another activity about the power management of nano nodes is the development of a user level interface to kernel power management features. The operating system provides access to the ARM specific power management and to the voltage regulators that supply the whole system. However, a user level interface to those features is needed to allow applications to tune their computational requirements and their power consumption. The development of such an interface was started and it is in its preliminary phase.

Last planned activity is the development of an activity profiler that allows to collect information about the whole system behaviour. Such data will allow to control the power consumption by selecting the most effective energy policy as function of the computational load and of the power supply status.

We started the development of a kernel scheduler augmentation, that will allow to expose information about running tasks and their resource requirements to a user space application.

## 3 Micro/Personal Node

The main research areas targeted by the micro/personal enabled node technologies are the following:

- Advanced biometric algorithms that are capable to identify the most significant features of the face and of the voice of a person.
- Access rights delegation for offline control systems.

### 3.1 Face and Voice Recognition for People Identification

The SPD technologies required by the “Face and voice recognition for people identification” usage scenario have been identified and analysed during the assessment phase and have been illustrated in the deliverable D3.1 “SPD node technologies assessment”. After the assessment phase, during the first part of the project, the effort has been concentrated on the study of the technologies related to face recognition. The results of this activity will allow the implementation of a first prototype that demonstrates the part of the “Face and voice recognition” usage scenario related only to face recognition. The prototype will be capable to identify a person comparing the photos acquired by a camera, the photo available on his/her identification document (identification card or passport) and a biometric profile database.

In the second part of the project the activity will focus on the other part of the usage scenario that is related to the identification of people based on voice recognition and verification. The final prototype will demonstrate the capabilities of a recognition embedded system based on multiple biometric sources, introducing in this way an increased level of security, privacy and dependability in the context of people identification.

#### 3.1.1 Embedded System Based Face Recognition for People Identification

Several new face recognition technologies have been proposed recently in the scientific community and they have been evaluated and assessed in D3.1 “SPD node technologies assessment”. The new technologies are based mainly on the following approaches:

- usage of three-dimensional (3D) scans,
- recognition from high resolution still images,
- recognition from multiple still images and
- multi-modal face recognition.

In addition, considering real applications, it is important to consider a set of multi-algorithms and pre-processing algorithms that can be applied to the previous approaches in order to correct the illumination and pose variations. The goal of these technologies is to significantly improve the performance of automatic face recognition, in particular when it is based on embedded system platforms.

The assessment phase performed during the first part of the project produced the following results:

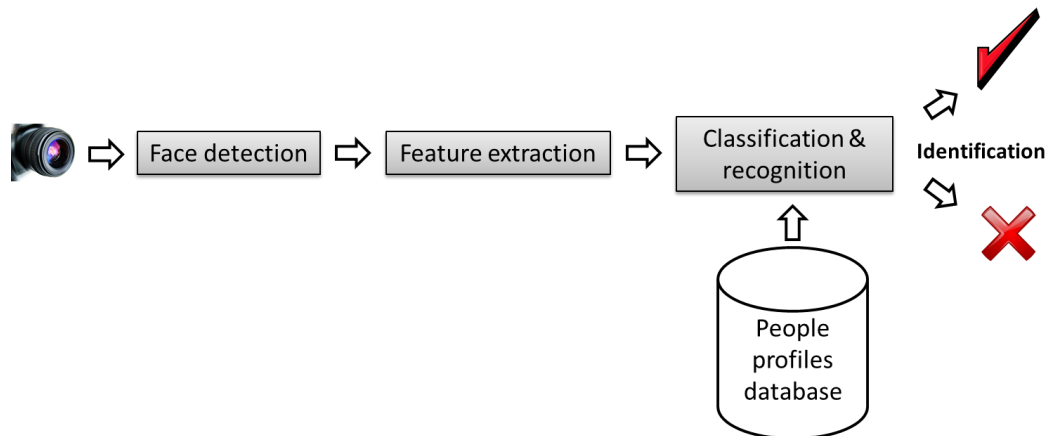
- identification of the most suitable recognition and identification algorithm,
- definition of the evaluation framework.

##### 3.1.1.1 The Recognition Algorithm

The approach identified in the assessment phase that satisfies the technical requirements for face recognition and that provides a contribution in terms of SPD capabilities is the Eigenface method. This method is based on the idea of extracting the basic features of the face: the objective is to reduce the problem to a lower dimension maintaining, at the same time, the level of dependability required for this application context. This approach has been theoretically studied during the nineties and has been recently reconsidered because, considering the ratio between the required resources and the quality of the results, it is well dimensioned for embedded systems, [20] and [21]. Today, it is becoming the most

credited method for face recognition, [22] and [23]. The core of this solution is the extraction of the principal components of the faces distribution, which is performed using the Principal Component Analysis (PCA) method. This method is also known in the pattern recognition context as Karhunen-Loève (KL) transform. The principal components of the faces are eigenvectors and can be computed from the covariance matrix of the face pictures set (faces to recognize). Every single eigenvector represents the feature set of the differences among the face picture set. The graphical representations of the eigenvectors are also similar to real faces and, for this reason, they are called eigenfaces. The PCA method is autonomous and therefore is particularly suggested for unsupervised and automatic face recognition systems.

The recognition process is performed conceptually in three steps:



**Figure 3-1: Recognition process main phases.**

The mathematical key concept on which the algorithm is based is that the eigenfaces are the principal components of the distribution of faces, or the eigenvectors of the covariance matrix of the set of face images. Starting from this idea the approach proposes the following steps:

1. The PCA method is used to find the eigen-vectors, called “eigenfaces”, of the covariance matrix corresponding to the generic training images.
2. The eigenvectors are ordered to represent different amounts of the variation, respectively, among the faces. Each face can be represented exactly by a linear combination of the eigenfaces. It can also be approximated using only the “best” eigenvectors with the largest eigenvalues.
3. Identification of the “face space”: the best M eigenfaces are used to construct an M dimensional space.
4. In the final step, unknown face pictures are projected on the face space to compute the distance from the reference faces. The weights describing each face are obtained by projecting the face image onto the eigenface.

The details of the algorithm and of its implementation are provided in the following sub-sections.

### 3.1.1.2 Identification of Eigenvectors and Eigenfaces

The eigenfaces set defines the so called “face space”. In the recognition phase, the unknown face pictures are projected on the face space to compute the distance from the reference faces.

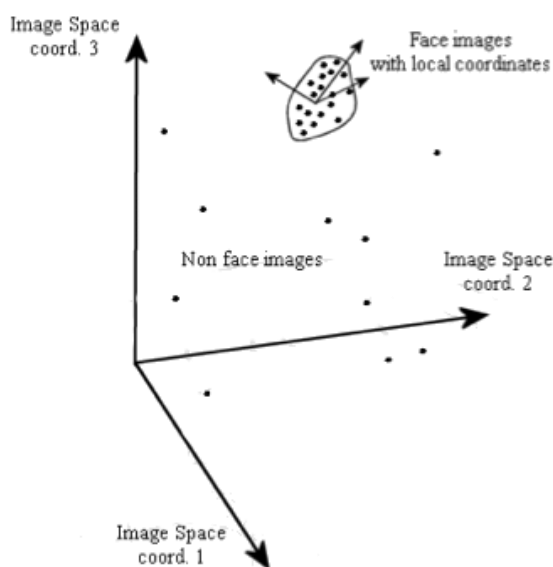
Each unknown face is represented (reducing the dimensionality of the problem) by encoding the differences from a selection of the reference face pictures. The unknown face approximation operation considers only the eigenfaces providing higher eigenvalues (variance index in the face space). In other words in the face recognition the unknown face is projected on the face space to compute a set of weights of differences with the reference eigenvalues. This operation allows to recognize if the picture contains a



face (known or not), that happens if its projection is close enough to the reference face space. In this case the face is classified using the computed weights, deciding for a known or unknown face. A recurring unknown face can be added to the reference known face set, recalculating the whole face space. The best matching of the face projection with the faces in the reference faces set allows identifying the individual.

Going in the detail of the “face space”, the evaluation process requires some introductory considerations. A generic bi-dimensional picture can be grey level converted and eventually adjusted for brightness and contrast. If square shaped (the general case slightly differs), it can be defined by an  $N \times N$  matrix of pixels, and each of them is a point in a  $N^2$ -dimension space. A set of pictures can hence map to a set of points on this space.

In our case, every picture refers to faces: the representation in the  $N^2$  space will not be randomly distributed. Also, the PCA analysis provides the best vectors representing the pictures distribution. It's possible to gather that these vectors can define a subspace (of the whole space) for generic face pictures (called “face space”). The following figure shows this concept.



**Figure 3-2: Space distribution of faces images.**

Each vector of the subspace so defined has a dimension  $N$ ; these vectors are the eigenvector of the covariance matrix corresponding to the original images, and given that shown have the appearance of a face, they are called "eigenface".

More formally, given a training set of images:

$$\Gamma_1, \Gamma_2, \dots, \Gamma_M$$

the average face is computed as:

$$\psi = \frac{1}{M} \sum_{i=1}^M \Gamma_i$$

Each face of the training set differs from the average according to the vector:

$$\phi_i = \Gamma_i - \psi$$

This set of vectors of large size is then subjected to analysis of the main components that allows to obtain a set of orthonormal vectors  $u_i$  and of scalars  $\lambda_i$  associated with them, that best describes the data distribution. The vectors  $u_i$  and the scalars  $\lambda_i$  are respectively the eigenvector and the eigenvalue of the covariance matrix:

$$C = \frac{1}{M} \sum_{i=1}^M \phi_i \phi_i^T = \frac{1}{M} A A^T \quad \text{where the matrix } A = [\phi_1 \phi_2 \dots \phi_M]$$

### 3.1.1.3 Complexity Reduction

The mechanism that allows reducing the complexity of the problem is based on the identification of the  $M_i$  ( $M_i \leq M$ ) further eigenvalue of the training set, with which it is possible to select the corresponding eigenvector. These form the basis of a new space of representation of data, particularly from the reduced dimensionality. The number of eigenvectors considered is chosen heuristically and depends strongly on the distribution of the eigenvalue. To improve the effectiveness of this approximation the background is normally cut from the images; in this way it makes zero the value of the eigenface outside of the face.

At this point the identification is a simple pattern-recognition process.

Every new image  $\Gamma$  to identify is transformed into the eigenface components through a projection on the "face space" with the simple operation:

$$\omega_k = u_k^T (\Gamma - \psi),$$

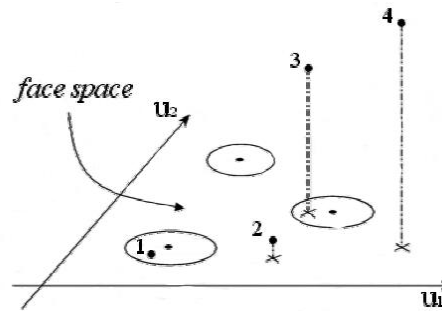
with  $k=1, \dots, M'$  (and  $u_k^T$  transposed to the base of the transformed space) that consists of multiplications and sums, point to point of the image.

The values thus obtained from a weight vector  $\Omega^T$  which expresses the contribution of each eigenface in representing the image data. It is now clear how  $M'$  eigenface may constitute a basis set to represent the other images. The vector is used to determine, if it exists, which of the predefined classes describes in best way the image (through a nearest-neighbour algorithm type). The simplest way to determine which class best describes the face in question consists in identifying the class  $k$  that minimizes the Euclidean distance.

A face is classified as belonging to the class  $k$  if the minimum distance  $e_k$  is below a predetermined threshold value  $\theta_\varepsilon$ ; otherwise the face is classified as unknown. In addition to this and to consider that the image of a generic face should project itself in extreme proximity of the "face space" that in general, as it was built (the faces of the training set), should describe all the images with the appearance of a face. In other words, the distance  $\varepsilon$  of an image from its projection should be within a certain threshold.

In general, four possible cases may arise, as shown in the figure below.

1. the carrier is near the "face space" and its projection to a class;
2. the carrier is near the "face space", but its projection is not close to any known class;
3. the carrier is far from the "face space", but its projection is close to a class known;
4. the carrier is far from the "face space" and its projection is not close to any class known.



**Figure 3-3: Example of a simple "face space" consisting of just two eigenfaces ( $u_1$  and  $u_2$ ) and from three individuals known ( $\Omega_1$ ,  $\Omega_2$  and  $\Omega_3$ ).**

The recognition algorithm reacts in four different ways:

1. In the first case, the individual is recognized and identified;
2. In the second case the algorithm detects only the presence of the individual but doesn't recognize him/her;
3. The third case could represent a typical false-positive, because of the apparent distance between the carrier and the "face space", and the algorithm can refuse to recognize the person;
4. Finally, in the fourth case it is assumed that the image doesn't contain a face.

The detection of faces within an image can be performed also using the space formed by the best eigenface. The creation of the weight vector is nothing but a projection of the space "facespace" low dimensional ( $\omega_k = u_k^T (\Gamma - \psi)$ ), so the distance  $\varepsilon$  between the image and its projection coincides with the distance between the image of the average deducted:

$$\varphi = \Gamma - \psi$$

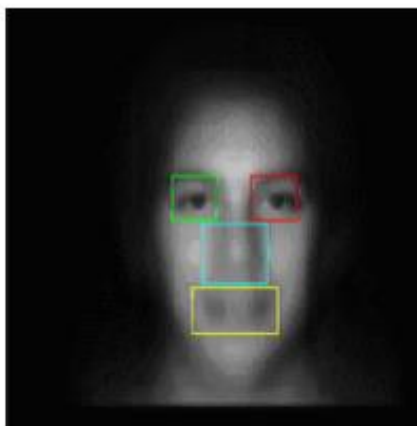
and the projection of the vector of weights in the "face space":

$$\phi_f = \sum_{k=1}^{M^i} \omega_k u_k$$

Note that in this case, the appearance of the projected image will not be in general any feature of the face. To detect the presence of a face in the image it is necessary to calculate the distances between different portions of the image and the projection on the face sought. In this way is to generate a map ("facemap") of distances  $\varepsilon(x, y)$ . The only flaw of this approach for the identification of faces is the computational cost, which increases with the granularity with which it analyses the image.

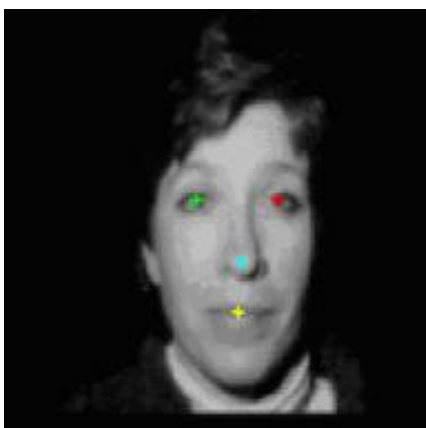
The next step is to try to extend the eigenface in order to make it well suited to managing large databases. We adopt the so-called "modular eigenspace" that, when used in support of traditional Eigenface, can demonstrate an improvement in recognition accuracy. This extension consists in an additional "layer" of key features of the face such as eyes, nose and mouth (as shown in the figure below). In this circumstance one speaks of: eigeneye, eigennose and eigenmouth, or more generally of eigenfeature.

The new representation of the faces can be seen, in a modular fashion, as a description of the entire low-resolution face, combined with a more detailed facial features on the most salient.



**Figure 3-4: Eigenface in which domains were identified: eigeneye (left and right), eigennose and eigenmouth.**

Of course achieving this technique needs an automated method of detection of the characteristic elements of the image (shown in the figure below): this can be taken from the mechanism adopted to identify faces offered directly from the Eigenface. Similarly to the distance from the space of faces, in this circumstance is called away from the “feature space”.



**Figure 3-5: Example of identification of eigenfeature.**

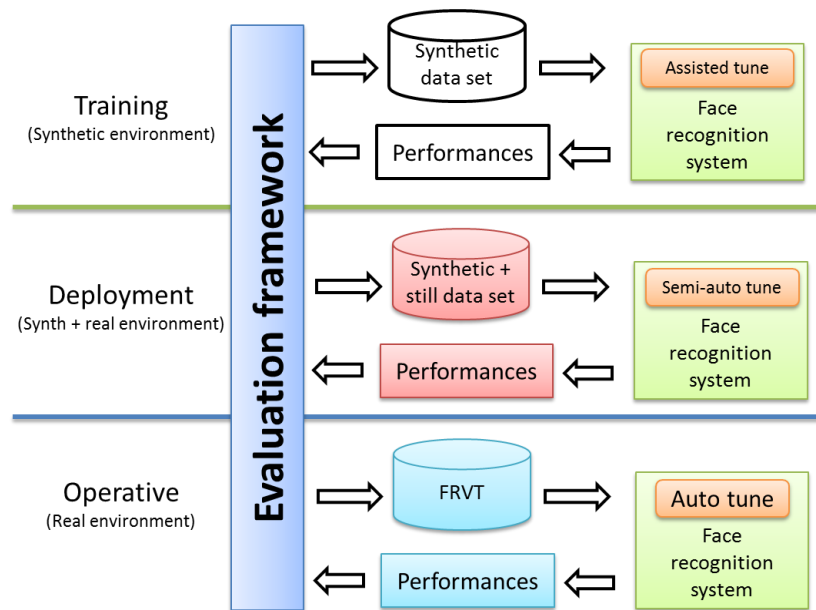
This extension is suitable above all to offer a valuable mechanism for modular reconstruction of images, which is advantageous in terms of compression. And thanks to the more details provided by eigenfeature the reconstructed images show a higher quality than the reconstruction from eigenface.

The advantage offered by the eigenfeature is the ability to overcome some weaknesses of the standard eigenface method. In fact, the standard eigenface recognition system can be fooled by gross variations in the input image (hats, beards, etc.).

Finally, the use of infrared images with the eigenface technique has been revealed successful. An infrared image (or thermogram) has the characteristic of showing the distribution of heat emitted by an object. While this approach may prove a formidable strength to attack with "mask", and the ability to work with any type of lighting (also absent), may ultimately prove a big problem with people who wear glasses, as in infrared these are very often completely opaque. Considering images of individuals without glasses the benefits are obvious, especially on profile pictures.

### 3.1.2 The Evaluation Framework

The evaluation of the quality and reliability of the recognition process is fundamental in order to guarantee the required levels of security and dependability. The evaluation can be considered conceptually as a part of the recognition algorithm itself, providing feedbacks that close the retroaction loop. This phase allows the system to adjust the parameters of the recognition algorithm, in order to maintain the maximum level of identification reliability. The problem addressed during the evaluation phase is very complex and plays an important role since from the beginning of the system configuration and deployment: during the initial training phase, a controlled biometric dataset is used to train the recognition algorithm and obtain the configuration that guarantee the minimum level of identification reliability. In this case, the evaluation process is used to understand when the system is ready for the deployment (as shown in the figure below).



**Figure 3-6: The evaluation framework in the recognition system life cycle.**

From a technical point of view, considering the complexity of the addressed problem, the evaluation process cannot be performed simply by the recognition algorithm but requires a complete evaluation framework that integrates with the recognition system. The evaluation framework is composed by:

- A standard evaluation data set: the Embedded Face Recognition System (EFRS) proposed in nSHIELD adopts a data corpus that must contain at least 50,000 recordings divided into training and validation partitions. The data corpus is constituted by high resolution still images, taken under controlled lighting conditions and with unstructured illumination, 3D scans and contemporaneously still images collected in a real environment.
- A challenging problem that allows the evaluation of the improvement in terms of performance: the identification of a challenging recognition scenario ensures that the evaluation is performed on sufficiently reasonable, complex and large problems and that the results obtained are valuable, in particular when compared between different configurations and recognition algorithms. The challenging problem identified to evaluate the EFRS consists of six experiments. The experiments measure the performance on still images taken with controlled lighting and background, uncontrolled lighting and background, 3D imagery, multi-still imagery, and between 3D and still images.
- A software evaluation infrastructure: it supports an objective comparison among different recognition configuration and algorithms. The infrastructure ensures that results from different algorithms are computed on the same data sets and that performance scores are generated with

the same protocol. To measure the improvements introduced by the EFRS and perform the run time evaluation required by the recognition system we selected the Face Recognition Vendor Test of year 2002 (FRVT) [24], that “provide independent government evaluations of commercially available and prototype face recognition technologies. These evaluations are designed to provide U.S. Government and law enforcement agencies with information to assist them in determining where and how facial recognition technology can best be deployed. In addition, FRVT results help identify future research directions for the face recognition community”. FRVT 2002 consists of two tests: the High Computational Intensity (HCInt) Test and the Medium Computational Intensity (MCInt) Test. Both tests require the systems to be full automatic, and manual intervention is not allowed.

### 3.1.2.1 Design of Data Set and Challenge Problem

The design of the EFRS starts from the performance measured using FRVT. It establishes a performance goal that is an order of magnitude greater than FRVT measured performance. Starting from this goal, it introduces a data corpus and a challenge problem that are significant and valuable for real application. The evaluation process, using this data corpus and challenge problem, aims at understanding if the EFRS has reached the FRVT goal and is capable to maintain at runtime this performance improvement.

The starting point for measuring the improvement of performance is the high computational intensity test (HCInt) of the FRVT. The images in the HCInt corpus are taken indoors under controlled illumination. The performance point selected as the reference is a verification rate of 80% (error rate of 20%) at a false accept rate (FAR) of 0.1%. This is the performance level of the top three FRVT 2002 participants. An order of magnitude improvement in performance that we expect from EFRS requires a verification rate of 98% (2% error rate) at the same fixed FAR of 0.1%.

A challenge for designing the EFRS is collecting sufficient data to measure an error rate of 2%. The verification performance is characterized by two statistics: verification rate and false acceptance rate. The false acceptance rate is computed from comparisons between faces of different people. These comparisons are called non-matches. In most experiments, there are sufficient non-match scores because the number of non-match scores is usually quadratic in the size of the data set. The verification rate is computed from comparisons between two facial images of the same person. These comparisons are called match scores. Because the number of match scores is linear in the data set size, generating a sufficient number of matches can be difficult.

For a verification rate of 98%, the expected verification error rate is one in every 50 match scores. To be able to perform advanced statistical analysis, 50,000 match scores are required. From 50,000 match scores, the expected number of verification errors is 1,000 (at the EFRS performance goal).

The challenge is to design a data collection protocol that yields 50,000 match scores. We accomplished this by collecting images for a medium number of people with a medium number of replicates. The proposed EFRS data collection is based on the acquisition of images of 200 subjects once a week for a year, which generates approximately 50,000 match scores.

The design, development, tuning and evaluation of the face recognition algorithms require three data partitions: training, validation, and testing. The EFRS challenge problem provides training and validation partitions to developers. A separate testing partition is being collected and sequestered for an independent evaluation.

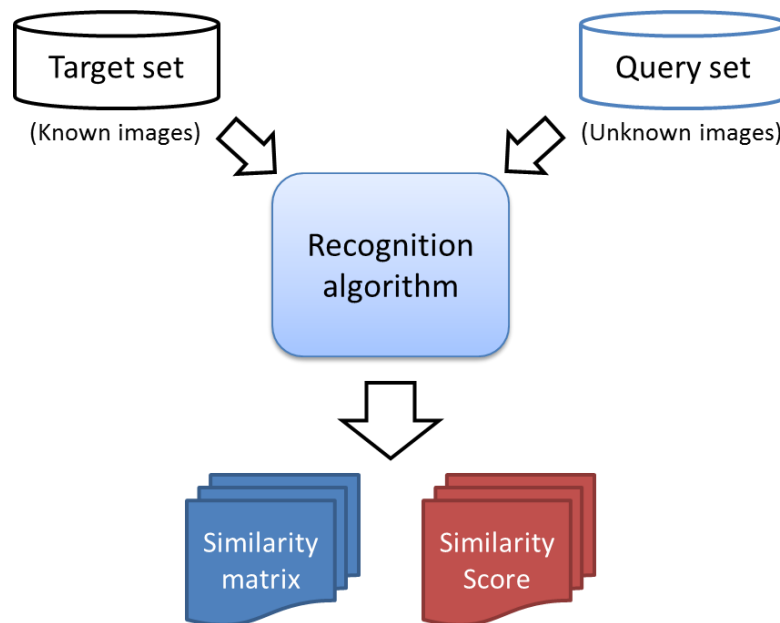
The representation, feature selection, and classifier training is conducted on the training partition. For example, in PCA-based (Principle Component Analysis) and LDA-based (Linear Discriminant Analysis) face recognition, the subspace representation is learned from the training set. In vector machine (SVM) based face recognition algorithms, the SVM classifier is trained on the data in the training partition.

The challenge problem experiments must be constructed from data in the validation partition. During algorithm development, repeated runs are made on the challenge problems. This allows developers to assess the best approaches and tune their algorithms. Repeated runs produce algorithms that are tuned

to the validation partition. An algorithm that is not designed properly will not generalize to another data set.

To obtain an objective measure of performance it is necessary that the results are computed on a separate test data set. The test partition measures how well an approach generalizes to another data set. By sequestering the data in test partition, participants cannot tune their algorithm or system to the test data. This allows for an unbiased assessment of algorithm and system performance.

The EFRS experimental protocol is based on the FRVT 2002 testing protocols. For an experiment, the input to an algorithm is two sets of images: target and query sets. Images in the target set represent facial images known to the system. Images in the query set represent unknown images presented to the system for recognition and identification. The output from an algorithm is a similarity matrix, in which each element is a similarity score that measures the degree of similarity between two facial images. The similarity matrix is comprised of the similarity scores between all pairs of images in the target and query matrices. Verification scores are computed from the similarity matrix (see next figure).



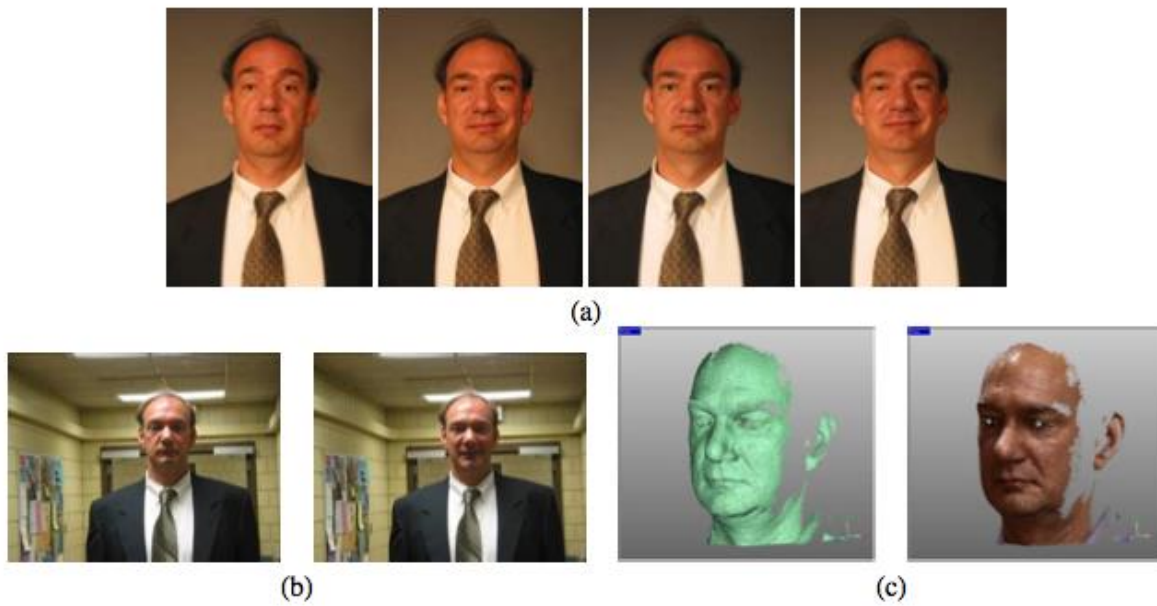
**Figure 3-7: The testing protocol.**

### 3.1.2.2 Description of the Data Set

The EFRS data corpus is part of an ongoing multi-modal biometric data collection.

A subject session is the set of all images of a person taken each time a person's biometric data is collected. The EFRS data for a subject session consists of four controlled still images, two uncontrolled still images, and one three-dimensional image. The figure below shows a set of images for one subject session. The controlled images are taken in a studio setting, are full frontal facial images taken under two lighting conditions (two or three studio lights) and with two facial expressions (smiling and neutral). The uncontrolled images were taken in varying illumination conditions; e.g., hallways, atria, or outdoors. Each set of uncontrolled images contains two expressions, smiling and neutral. The 3D images are taken under controlled illumination conditions appropriate for the sensor (structured light sensor that takes a 640 by 480 range sampling and a registered colour image), not the same as the conditions for the controlled still images. In the FRP, 3D images consist of both range and texture channels. The sensor acquires the texture channel just after the acquisition of the shape channel. This can result in subject motion that can cause poor registration between the texture and shape channels. The still images are taken with a 4 Megapixel camera.





**Figure 3-8: Images from one subject session. (a) Four controlled stills, (b) two uncontrolled stills, and (c) 3D shape channel and texture channel pasted on 3D shape channel.**

**Table 3-1: Size of faces in the validation set imagery broken out by category. Size is measured in pixels between the centres of the eyes. The table reports mean, median, and standard deviation.**

	Mean	Median	Standard Deviation
<b>Controlled</b>	261	260	19
<b>Uncontrolled</b>	144	143	14
<b>3D</b>	160	162	15

Images are either 1704x2272 pixels or 1200x1600 pixels, in JPEG format and storage sizes range from 1.2 Mbytes to 3.1 Mbytes. Subjects have been photographed approximately 1.5 meters from the sensor.

The table above summarizes the size of the images for the uncontrolled, controlled, and 3D image categories. The average distance between the centres of the eyes in the FRVT 2002 database is 68 pixels with a standard deviation of 8.7 pixels: the data set adopted for the EFRS satisfies the FRVT 2002 and provide a quality largely better than the FRVT 2002. This means that we follow this standard but we use a source of information that makes the test at least 4 to 6 times harder.

The data required for the experiments on the EFRS are divided into training and validation partitions. From the training partition, two training sets are distributed. The first is the large still training set, which is designed for training still face recognition algorithms. The large still training set consists of 12,776 images from 222 subjects, with 6,388 controlled still images and 6,388 uncontrolled still images. The large still training set contains from 9 to 16 subject sessions per subject, with the mode being 16. The second training set is the 3D training set that contains 3D scans, and controlled and uncontrolled still images from 943 subject sessions. The 3D training set is for training 3D and 3D to 2D algorithms. Still face recognition algorithms can be training from the 3D training set when experiments that compare 3D and still algorithms need to control for training.

The validation set contains images from 466 subjects collected in 4,007 subject sessions. The demographics of the validation partition broken out by sex, age, and race are given in the following figure.



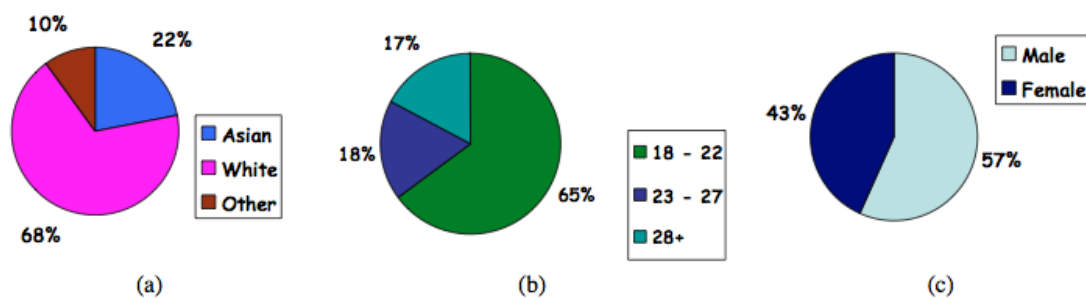


Figure 3-9: Demographics of FRP ver2.0 validation partition by (a) race, (b) age, and (c) sex.

The validation partition contains from 1 to 22 subject sessions per subject (see figure below).

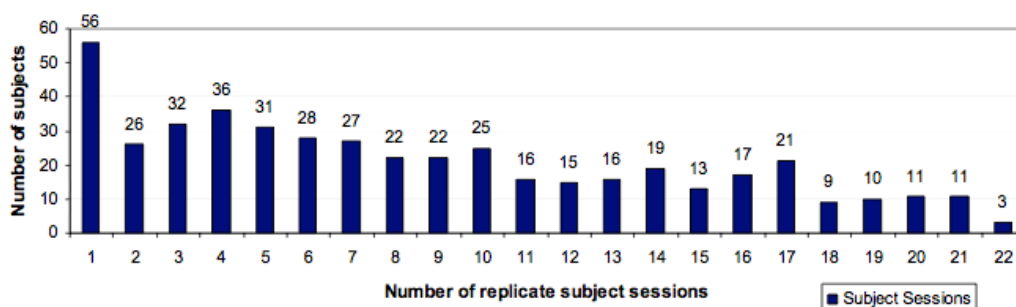


Figure 3-10: Histogram of the distribution of subjects for a given number of replicate subject sessions. The histogram is for the ver2.0 validation partition.

### 3.1.2.3 Description of the Evaluation Test

The experiments that will be performed to evaluate the EFRS are designed to improve the face recognition algorithm with emphasis on 3D and high resolution still imagery. EFRS will perform six tests:

1. The first test measures the performance on the classic face recognition problem that is the recognition from frontal facial images taken under controlled illumination. To encourage the development of high resolution recognition, all controlled still images are taken in high resolution. In this test, the biometric samples in the target and query sets consist of a single controlled still image. It is clear that multi-still images of a person can substantially improve performance. This test operates in 2D.
2. The second test is designed to examine the effect of multiple still images on performance. In this test, each biometric sample consists of the four controlled images of a person taken in a subject session. The biometric samples in the target and query sets are composed of the four controlled images of each person from a subject session. This test operates in 2D.
3. The third test is focused on 3D imagery and measures performance when both the enrolled and query images are in 3D. In this test, the target and query sets consist of 3D facial images. One potential scenario for 3D face recognition is that the enrolled images are 3D and the target images are still 2D images.
4. The fourth test is designed to measure the progress on recognition from uncontrolled frontal still images. In this test, the target set consists of single controlled still images, while the query set consists of single uncontrolled still images. The “supporters” of 3D face recognition claim that 3D imagery is capable of achieving an order of magnitude of improvement in face recognition performance. Recognizing faces under uncontrolled illumination has numerous applications and is one of the most difficult problems in face recognition.

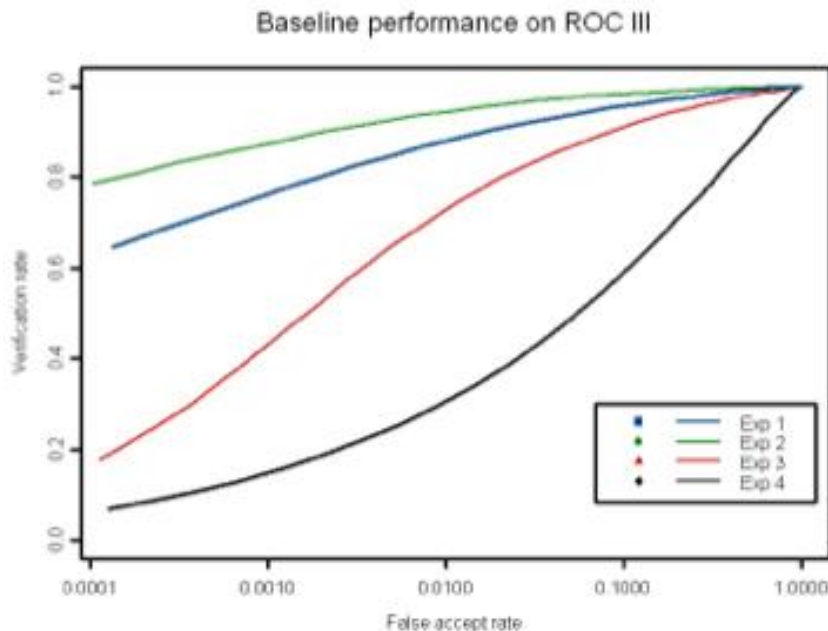
5. The fifth test explores this scenario when the query images are controlled. The query set consists of a single controlled still. This test operates on 3D Imagery.
6. Finally, test number six examines the uncontrolled query image scenario. The query set consists of a single uncontrolled still and the test operates on 3D Imagery.

#### 3.1.2.4 Baseline Performance

The baseline performance is introduced to demonstrate that a challenge problem can be executed, can provide a minimum level of performance and a set of controls/feedback for detailed studies and evaluation. The face recognition algorithm based on PCA has been selected as the baseline algorithm.

The initial set of baseline performance results has been provided for test 1, 2, 3, and 4. For test 1, 2, and 4, baseline scores have been computed from the same PCA-based implementation. In test 2, a fusion module is added to handle multiple recordings in the biometric samples. The algorithm is trained on a subset of 2,048 images from the large training set. The representation consists of the first 1,228 eigenfeatures (60% of the total eigenfeatures). All images were pre-processed by performing geometric normalization, masking, histogram equalization, and rescaling pixels to have mean zero and unit variance. All PCA spaces have been whitened. The distance in nearest neighbour classifier is the cosine of the angle between two representations in a PCA-space. In test 2, each biometric sample consists of four still images, and comparing two biometric samples involves two sets of four images. Matching all four images in both sets produces 16 similarity scores. For test 2, the final similarity score between the two biometric samples is the average of the 16 similarity scores between the individual still images.

An example set of baseline performance results is given for test 3 (2D versus 3D face recognition) in the following paragraphs. It has been obtained in the previous test performed by independent research team and can be considered as a reference point. The baseline algorithm for the 3D scans consists of PCA performed on the shape and texture channels separately and then fused. Performance scores are given for each channel separately and for the shape and texture channels fused. We also fused the 3D shape channel and one of the controlled still images. The controlled still is taken from the same subject session as the 3D scan. Using the controlled still models a situation where superior still camera is incorporated into the 3D sensor. The baseline algorithm for the texture channel is the same as in test 1.

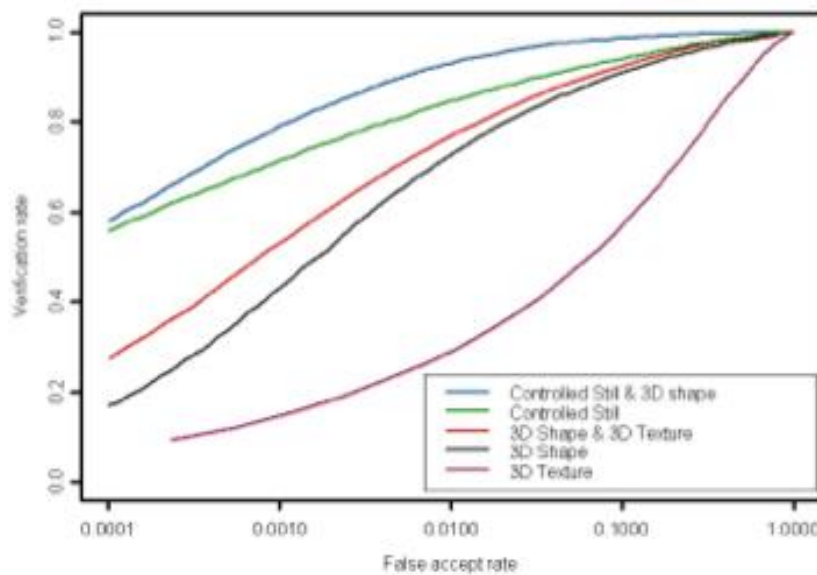


**Figure 3-11: Example of expected baseline ROC performance for test 1, 2, 3, and 4.**

The PCA algorithm adapted for 3D is based on Chang et al. [25].

The results obtained in the example of baseline verification performance for test 1, 2, 3, and 4 are shown in the above figure. Verification performance is computed from target images collected in the fall semester and query images collected in the spring semester. For these results, the time lapse between images is between two and ten months. The performance is reported on a Receiver Operator Characteristic (ROC) that shows the trade-off between verification and false accepts rates. The false accept rate axis is logarithmic. The results for test 3 are based on fused shape and texture channels. The best baseline performance should be achieved by multi-still images, followed by a single controlled still, and then 3D scans. The most difficult category should be the uncontrolled stills.

The figure below shows another example of baseline performance for five configurations of the 3D baseline algorithms: fusion of 3D shape and one controlled still; controlled still; fusion of 3D shape and 3D texture; 3D shape; and 3D texture. The best result is achieved by fusing the 3D shape channel and one controlled still image. This result suggests that 3D sensors equipped with higher quality still cameras and illumination better optimized to still cameras may improve performance of 3D systems.

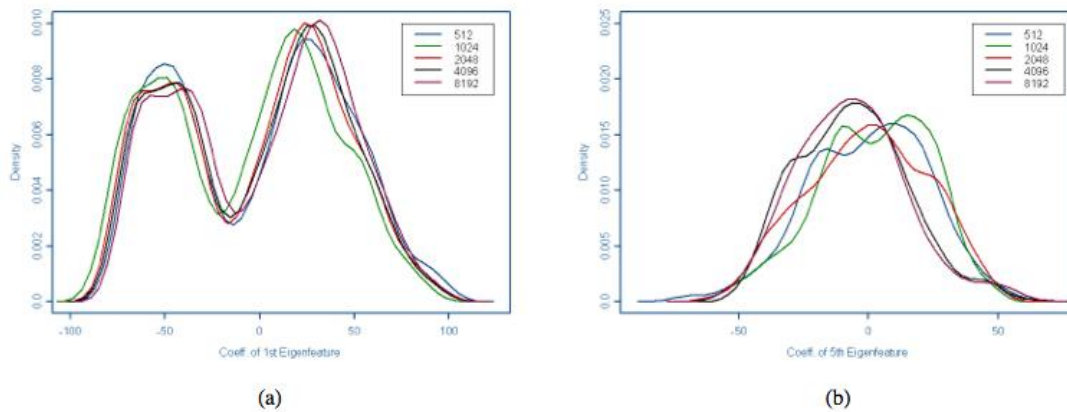


**Figure 3-12: Example of baseline ROC performance for Experiment 3 component study.**

Successful development of pattern recognition algorithms requires that one knows the distributional properties of objects being recognized. A natural starting point is PCA, which assumes the facial distribution has a multi-variate Gaussian distribution in projection space.

In the first facial statistics experiment we examine the effect of the training set size on the eigenspectrum. If the eigenspectrum is stable, then the variance of the facial statistics on the principal components is stable. The eigenspectrum is computed for five training sets of size 512, 1,024, 2,048, 4,096, and 8,192. All the training sets are subsets of the large still training set. The expected eigenspectra should be similar to the ones plotted in the figure below. The horizontal axis is the index for the eigenvalue on a logarithmic scale and the vertical axis is the eigenvalue on a logarithmic scale. The main part of the spectrum consists of the low to mid order eigenvalues. For all five eigenspectra, the main parts overlap.

The eigenvalues are estimates of the variance of the facespace distribution along the principal axes. The figure below shows that the estimates of the variances on the principal components should be stable as the size of training set increases, excluding the tails. The main part of the eigenspectrum is approximately linear, which suggests that to a first order approximation there is a  $1/f$  relationship between eigen-index and the eigenvalues.



**Figure 3-13: Estimated densities.**

The figure above describes an example of performance on test 1 for training sets of size 512, 1,024, 2,048, 4,096, and 8,192. The figure illustrates the estimated densities for the (a) 1st and (b) 5th eigen-coefficients for each training set (the numbers in the legend are the training set size). To generate the curve label 1024 in (a), a set of images are projected on the 1st eigenfeature generated the 1024 training set. The set of images projected onto the eigenfeatures is a subset of 512 images in common to all five training sets. All other curves were generated in a similar manner. Verification performances at a false accept rate of 0.1% is reported (vertical axis). The horizontal axis is the number of eigenfeatures in the representation. The eigenfeatures selected are the first  $n$  components. The training set of size 512 approximates the size of the training set in the FERET Sep96 protocol. This curve approximates what was observed by Moon and Phillips [26], where performance increases, peaks, and then decreases slightly. Performance peaks for training sets of size 2,048 and 4,096 and then starts to decrease for the training set of size 8,192. For training sets of size 2,048 and 4,096, there is a large region where performance is stable. The training sets of size 2,048, 4,096, and 8,192 have tails where performance degrades to near zero.

The examples described in this section allowed to identify and demonstrate the two most important consequences that we expected from the experiments: first, it is evident that increasing the training set increases also the performance of the recognition, and second, it is clear that the selection of the cut off index is not critical.

## 3.2 Access Rights Delegation

Within the scope of 3.2 (WP3, Task 2) an approach to delegation of access rights has been investigated. In a network of offline trusted embedded systems, a node need to be able to authenticate another node requesting some privileges, but also to determine what – if any – privileges should be granted. A model for doing this has previously been developed by the project partner (Telcred), but this model assumes that all access rights are issued by a central trusted authority and does not support delegation.

A real world example where delegation of access rights would be relevant is where the access rights issued by the central authority are valid only if the node seeking access has first interacted with another node. For example node: “Allow access to door B, but only if the visitor has first passed door A”. In this example, door A is entrusted to verify that a correct passage has taken place.

The approach that was investigated uses a construct called a path array. Nodes to which some authority has been delegated can read and write to the path array, thereby proving that a bona fide interaction has taken place. This information can then be used by the next node in the path array as part of the access control decision.

The work was mainly carried out as a M.Sc. thesis at KTH, the Royal University of Technology in Stockholm.

### 3.2.1 Problem Statement

In an offline PACS (Physical Access Control System), there is no continuous exchange of information to verify and allow a user through a series of doors, whereas this is a common feature in an online PACS. Current offline systems are unable to force a user to follow a certain designated route, e.g. Room A should be accessed before entering room B. This project explores a model to enforce such a route, by using delegation of some authority from the main administrative system to the offline locks.

### 3.2.2 The Concept of “Path Array”

The developed artefact consists of a construct known as Path Array aka PA. Path Array is an array that can be one or multi-dimensional based upon the administrator requirements. PA consists of `Lockid` stored into each index of the array that needs to be accessed by the user in a sequence. Administrator is responsible to implement path array onto the user’s smart card before handing it over to the user.

`Ticket` that is stored in the flash memory of the smart card contains the PA. After the formation of mutual trust between the `Lock` and `Card`, `Lock` makes use of the remaining contents inside the `Ticket` for decision making.

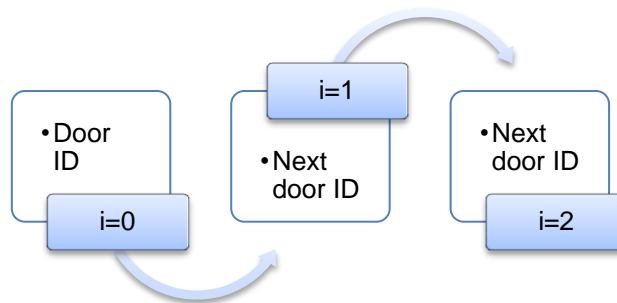


Figure 3-14: Path Array Design

The Figure above shows the outline of PA. PA consists of `Lockid` stored at each index ( $i = 0, 1, 2\dots$ ). PA holds the `Lockid` that should be accessible by a user in a sequence. `Server` digitally signs the PA stored inside the `Ticket`. Index  $i$  value starts from 0 and increments each time a user passes a door. This index value points to the `Lockid` that the user needs to visit. Hence, the contents of the `Ticket` will be as follows.

Unique Ticket ID			
Ticket Validity			
Access Policies			
Groups	Validity	Path array	Action
Client/Card Public Key			
Server Signature			

Figure 3-15: Ticket along with Path Array

### 3.2.3 Mechanism of the Artefact

After the creation of trust between the entities of offline PACS [27], `Lock` now processes the contents of PA, and then checks for its own ID at the current index  $i$ , if it is found then `Lock` performs three steps as follows,

- Increment index  $i$
- Generate HMAC and write it to `Card`
- Grant access to the user

If the `Lockid` present at index  $i$  does not correspond to `Lock` own id, it then it denies the access and logs the user action.

#### 3.2.3.1 Incrementing Index $i$

The path array PA contains lock ids stored inside it. Only the relative matching `Lock` is allowed to increment  $i$  value by one. At the time of generation of `Ticket` by the `Server`, it also generates a HMAC to be used by the first lock in the PA. The `Lock` located at the first index of PA makes use of this HMAC to ensure that no illegal modifications are done on the smart card. The Index of PA starts from the value 0. For instance, consider the below path array. This path array consists of lock ids B, A and C which should be followed in that order by the user.

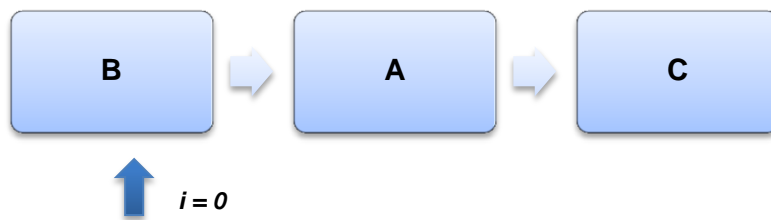


Figure 3-16: Ticket Incrementing the index value

In the figure above, the current value is 0 and `PA[0]=B`. Only the lock with id 'B' can increment the  $i$  value further.

#### 3.2.3.2 Generating Hash using HMAC

`Lock` creates the HMAC after incrementing the  $i$  value. HMAC stands for Hash Based Message Authentic Code. It calculates the message authentic code using a cryptographic hash function and shared secret key. In the offline PACS scenario, geographically dispersed locks securely exchange the messages among them by using message digest. HMAC is necessary in offline PACS scenario to ensure the integrity of smart card contents. The process of creating HMAC is as shown in the formula below.

$$HMAC(K, m) = H((K \oplus opad) \parallel H(K \oplus ipad) \parallel m)$$

where:

- $K$  is the shared secret key
- $m$  is the message to be protected
- `opad` is outer padding (0x5c5c....)
- `ipad` is inner padding (0x3636....)
- $H$  is the cryptographic hash function (MD5, SHA etc.)
- `||` is concatenation
- $\oplus$  is the exclusive-OR operation



The locks inside the facility were pre-installed with  $key_{shared}$ . Concatenating the  $key_{shared}$  with  $Lock_{id}$  generates the secret key  $key_{secret}$ . Message  $m$  in this context indicates the current index  $i$  value, and the rest of them use default parameters.

While hash generation,  $key_{secret} = key_{shared} || Lock_{id}$ .

Generation of HMAC is as following:

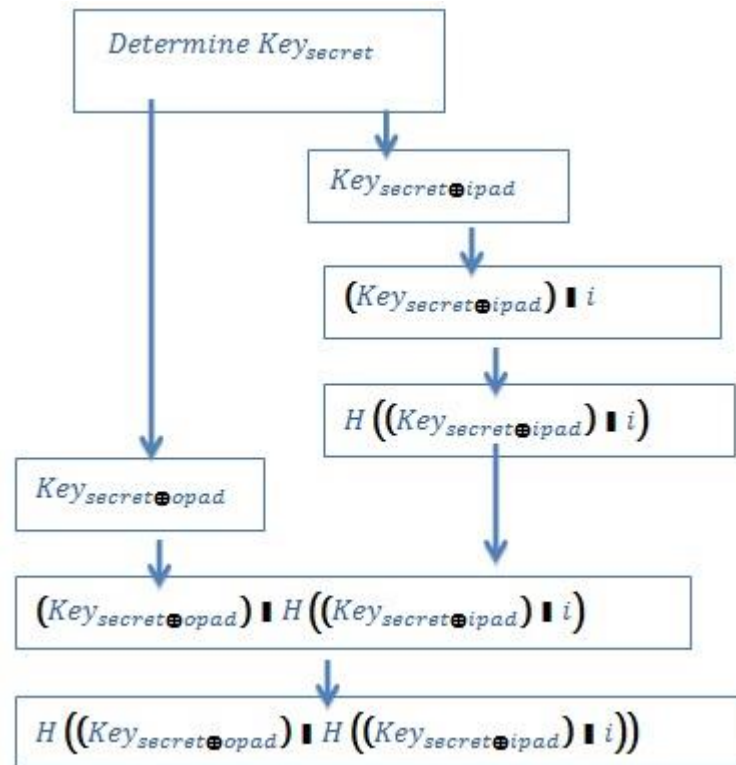


Figure 3-17: Process of HMAC creation

In the above figure,  $H$  can be any hash algorithm.

### 3.2.3.3 Generating Hash by the Locks in the Sequence

The overall concept of the artefact is to enforce the user through the path specified by the administrator. The user should attend this path in sequence. Hence, if one lock grants access to the user, which is present at index zero of PA i.e.,  $PA[0]$ , then the next lock in the sequence, which is available at  $PA[1]$  should be able to prove that the user has already passed through the door mentioned at  $PA[0]$ . Verification of the hash value generated by the lock present at  $PA[0]$  solves the above issue.

When the user presents his smart card at a  $Lock$  mentioned in the PA, the lock allows access only if it has confirmed that the user has already been allowed access by an earlier lock. During the verification process of HMAC  $Lock$  always uses the previous index for key generation and while in a hash generation process it uses its own  $Lock_{id}$ . Current lock initially checks  $i$  value.

#### 3.2.3.3.1 Scenario 1: If $i=0$

Then the lock knows that it is the first lock in the order. It then checks whether the value present at  $PA[0]$  matches its own lock id. If the id is not equal to its own id, it will log the user activity and deny the access.

Lock has the authority to perform further actions if the value at PA[0] matches its own id. It then verifies the HMAC stored by the *Server* on the *Card*, to make sure that nothing has been changed illegally. It will then increment *i* value by one and will generate HMAC by using the secret key  $key_{secret}$ . In this scenario,  $key_{secret}$  results from concatenating the  $key_{shared}$  with its own  $Lock_{id}$ .

### 3.2.3.3.2 Scenario 2: If $i > 0$

If *i* value is greater than zero, then lock confirms that the user has already accessed some doors in the sequence. Hence, it will confirm its own authority that it can change the contents of the card by looking up for its own lock id, and then generates hash to verify the hash stored by the earlier lock. Now, the *Lock* increments *i* value by one and generate a new hash to be used by the next lock in the series.

Verification steps by current lock in action are as follows,

- Step 1: reads current *i* value
- Step 2: Looks up present at PA[*i*]
- Step 3: If the value of own = PA[*i*], then proceed to step 4 else go to step 10
- Step 4: Verify the HMAC hash stored on smart card (generated by previous lock)
- Step 5: If the hash can be verified, continue else go to step 10
- Step 6: Increment *i* value by one
- Step 7: Generate new HMAC
- Step 8: Replace the old HMAC with generated HMAC to be used by next lock
- Step 9: Allow access and stop
- Step 10: Deny access and log user activity onto the card

Using above procedure the  $n^{th}$  lock will verify that the user has accessed the  $(n-1)^{th}$  lock, and this process continues with all the locks.



## 4 Power Node

The main research areas targeted by power node technologies are the following:

- Avionic system.
- Audio based surveillance infrastructure.
- Integration of heterogeneous embedded systems.
- GPU accelerated hashing and hash lookup.

### 4.1 Avionic System

#### 4.1.1 Description

SES has been working on designing an innovative dependable Avionic Architecture (OMNIA) to be employed as nSHIELD demonstrator. OMNIA has been built on SHIELD concepts and on an innovative avionic standard, following the rules of the Integrated Modular Avionic (IMA) in order to implement the IMA2G architecture: OMNIA is an Open System HW/SW architecture based on the IMA concept connected with “standard” in order to maximise the benefits.

The avionic applications are strongly driven by regulations and by a complex certification processes that increase the system design complexity, the deployment and the maintenance as well. Albeit, these processes of certification and regulation are expansive, they are mandatory and necessary to ensure an adequate level of security and dependability. The proposed architecture, conceptually based on SHIELD methodologies and on IMA, aims to simplify the overall processes of maintenance, certification and design, without jeopardizing the dependability and safety levels required by the domain.

The HW/SW units developed in accordance to the IMA are connected by a High Speed Serial Network based on the ARINC 664p7 standard.

Due to pre-existing avionic standards and solutions, SHIELD methodology will be modelled and embodied into the foregoing solution (IMA).

In the following subsections the OMNIA architecture, IMA based, is described.

#### 4.1.2 Architecture Modules and Interfaces

##### 4.1.2.1 System Overview

The architecture developed for this demonstrator will be composed of a network of Aircraft & Mission Management Computers (AMMC) and related Remote Interface Unit (RIU or NSIU) connected between them from a High Speed deterministic serial line, each “unit” is connected to the A/C (AirCraFt) sensors.

The system/network, by means of an additional middleware layer built around the RTPS architecture will virtualize the connection of a related sensor with all “computer units” that will be present in the system, allowing also the fault tolerance functionalities.

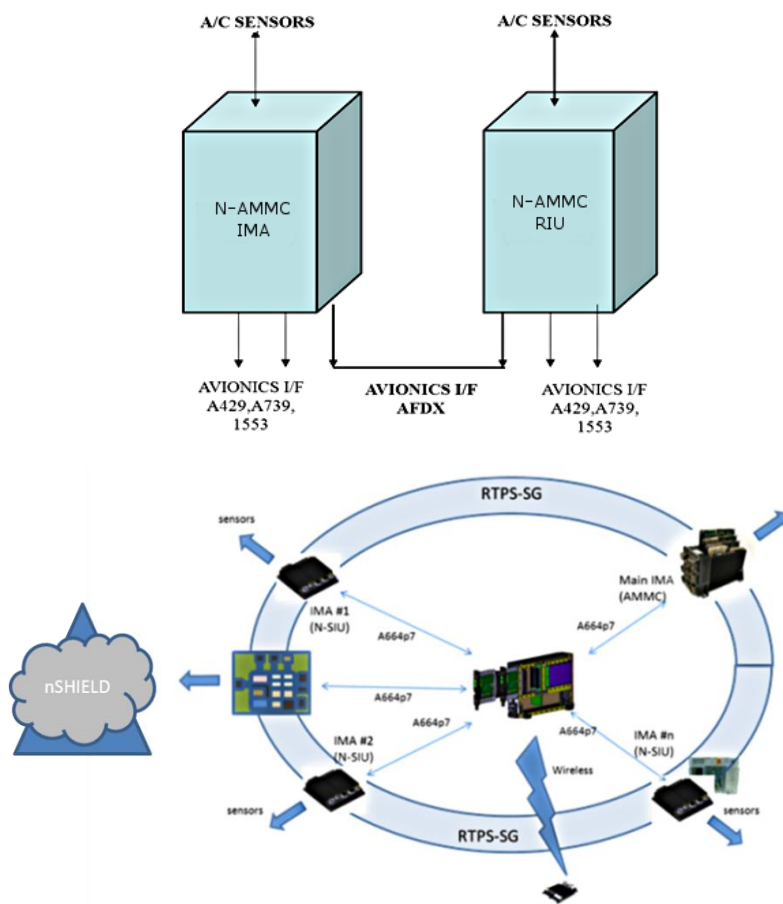
In particular:

- The main unit AMMC is used as IMA Central Unit.
- The other “units” (referenced as RIU or NSIU) are mainly used as sensor interfaces.
- IMA central unit and RIU-IMA are both “computer units”.
- All the “Computer Units” are connected via Ethernet (Rate constraint or Best effort methodology) among them, each “computer unit” can implement the interface with the avionic sensor.

The RTPS functionality has been integrated, as a library, in the Equipment Software (EQSW) environment. This library, according to the IMA concept, is segregated in a partition in order to increase the reliability and flexibility of the system. The RTPS represents the communication pattern used by the IMA and the RIU to exchange data. In particular, the publish-subscribe architecture is designed to simplify one-to-many data-distribution requirements. In this model, an application “publishes” data and “subscribes” to data. Publishers and subscribers are decoupled from each other too. Real-time applications require more functionalities than those provided by the traditional publish-subscribe semantics. The RTPS protocol adds publication and subscription timing parameters and properties so that the application developer can control different types of data flows and therefore the application’s performance and reliability goals can be achieved.

The RTPS developed has been implemented on the top of the UDP/IP protocol and tested on Ethernet; the determinism of the Ethernet is provided by the AFDX network.

In the figure below it is represented a logical view of the Demonstrator architecture.



**Figure 4-1: OMNIA Architecture.**

Exchanged data between all the IMA “computer units” could be:

- Discrete signals.
- Bus1553 data words.
- ARINC 429 data words
- In general all the I/O signals.

In particular, data are managed by the RIU and sent/received by the IMA Central Unit.

#### 4.1.2.2 Hardware description

In this section hardware components utilized for the demonstrator will be described.

As already mentioned, the solution developed by Selex Es is characterized by two N-AMMC equipments connected to each other via AFDX, where:

The IMA Central Unit is constituted mainly by a rack with:

- a processor module based on the PPC microprocessor (APM460);
- an Ethernet mezzanine card.

The RIU is constituted by a rack with:

- a processor module based on the PPC microprocessor (APM460);
- an Ethernet mezzanine card;
- two I/O boards (RS422, Arinc429, discrete and analog) (DASIO);
- a 1553 mezzanine card.

The following figure shows the hardware block diagram of the RIU configuration. Being a modular architecture, every component has been developed according to the actual avionic standards in terms of processing cycles, data buses, signal types, memory use, etc.

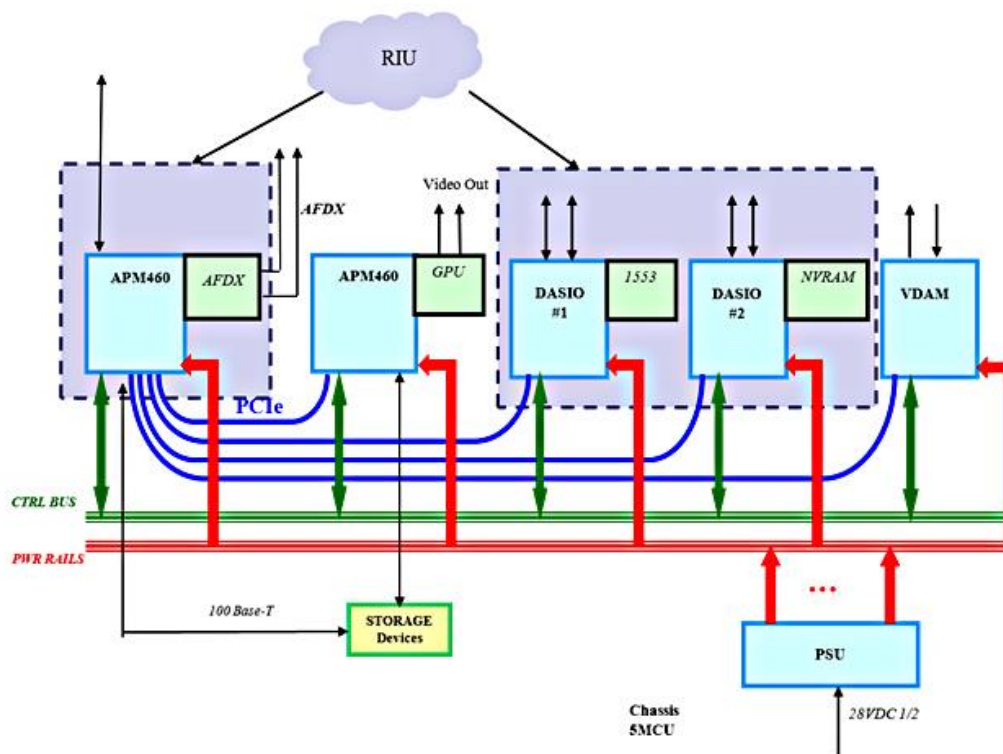


Figure 4-2: RIU HW block diagram.

#### 4.1.2.3 Software description

Regarding the software, each “computer unit” of the demonstrator is constituted by the following different components:

- RTPS SW: that provides the management of the RTPS protocol and the interface to application software layer (RTPS API).
- VMS (Vehicle management System) SW: that is constituted by the Resident SW and Equipment SW (EQSW).

These components run on the main processor. Also a software resident on the I/O boards is present.

The VMS SW can be divided in three layers:

- Application Layer;
- Middleware Layer;
- Module Support Layer.

More specifically, the Middleware layer is constituted by EQSW (API and Virtual Device Drivers) and RTPS SW, while the Module layer is represented by the Kernel between the Middleware and the physical layer. This architecture is represented in the following figure.

A space partitioning policy has been implemented. Each Application SW operates in one partition, while the RTPS SW operates in another partition, different from the first.

Communications in RTPS occur in three steps:

- Publisher declares intent to publish a publication.
- Subscriber declares interest in a publication.
- Publisher sends a publication issue.

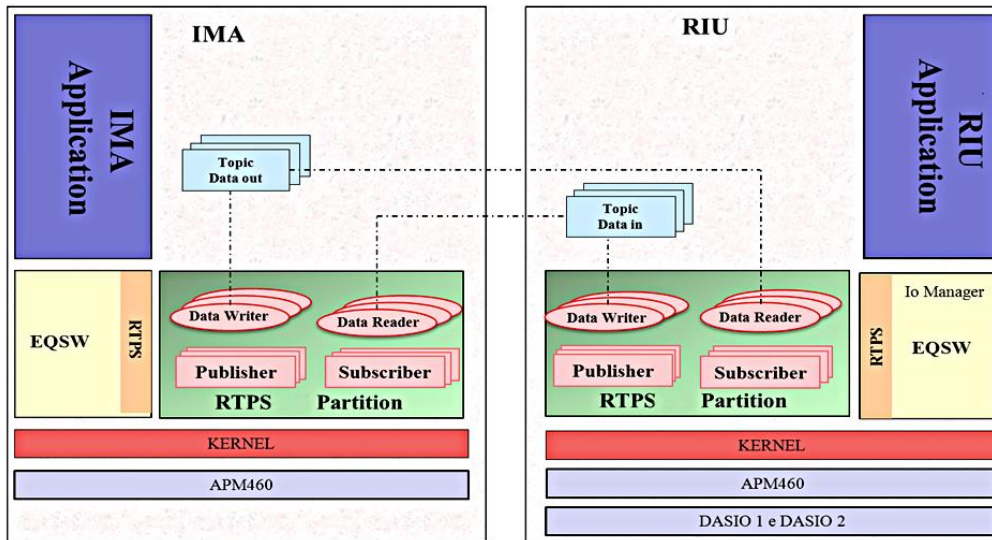


Figure 4-3: APM460SW.

In the figure above it is represented the software layers view where the IMA Application publishes Data\_out and subscribes Data\_in and, respectively, the RIU Application publishes Data\_in and subscribes Data\_out.

More in detail, the demonstrator read/write data via the DASIO boards of the RIU as described hereafter:

- The RIU Application reads the input discrete signals using the Discrete API provided by the EQSW;
- The RIU Application publishes the read data via the RTPS API;

- The IMA Application subscribes the input discrete data via the RTPS API.

Vice versa:

- The IMA Application publishes the output discrete signals via the RTPS API;
- The RIU Application subscribes the data to be written via the RTPS API;
- The RIU Application sends received data to the DASIO via the Discrete API.

According this description, the middleware handles three basic programming chores:

- Maintain the database that maps publishers to subscribers resulting in logical data channels for each publication between publishers and subscribers.
- Serialize (also called marshal) and de-serialize (or de-marshal) the data on its way to and from the network to reconcile publisher and subscriber platform differences.
- Deliver the data when it is published.

### 4.1.3 Metrics

In order to implement metrics described in the D2.5, the RTPS development will satisfy the following constraints:

- use of a BSP (Board Support Package), associated to the used board;
- use of an UDP/IP library;
- use of an RTOS for certifiable applications (i.e., GHS Integrity 178B).

The dependability of the system communication is guaranteed by the AFDX network that provides a dual link redundancy and Quality of Service. Besides, thanks to its topology and structure AFDX significantly reduces wires improving the system reliability.

The composability functionality will be provided thanks to the integration in the IMA architecture of a new module developed in accordance to the nSHIELD methodology and the ARINC standards.

## 4.2 Audio Based Surveillance

### 4.2.1 Description

#### 4.2.1.1 Introduction

ISD has been designing a novel audio based surveillance system that aims to overcome the most important limitations of non-military grade systems currently utilized in acoustic based research by providing correlated data acquisition from a large number of overlapping sensors.

Its hardware synchronized sensors make it ideal for applications requiring 3D sound capture such as acoustic hologram generation in real time. It features an extremely flexible and high performance DMA engine able to perform any type of spatial processing purely in hardware. It will be the only system able to deliver to main memory synchronized uncompressed audio streams from up to 768 microphones with zero CPU load. Its hierarchical structure is fully extensible and able to support any number of microphones. The targeted implementations will support from 8 up to 768 sensors in multiples of 8 units.

#### 4.2.1.2 System Boards

The system consists of 3 types of boards, namely the audio daughterboards, the concentrator boards and the grabber board.

The audio daughterboard, which has already been manufactured and tested, is used in order to receive audio streams from up to eight microphones, perform amplification of the audio signals and analogue to

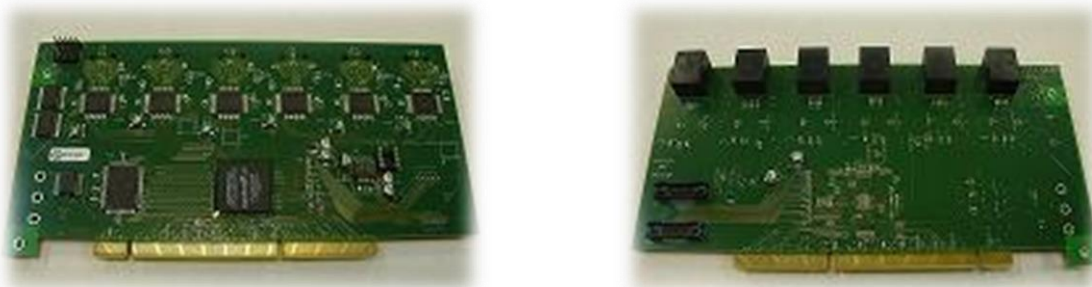
digital conversion. It supplies the combined digital audio stream to the concentrator board, on which it is plugged.



**Figure 4-4: The audio daughterboard**

The concentrator board is equipped with 16 input slots, where audio daughter boards can be plugged into. It is also equipped with a single output UTP port, able to deliver the combined audio streams to the grabber board. The concentrator board design is currently in progress.

Finally, the PCIX grabber board is equipped with 6 input UTP ports where concentrator boards can be connected using standard UTP cat6 network cables. It receives the streams and delivers the data to the host utilizing an extremely powerful and flexible DMA engine.



**Figure 4-5: The grabber board**

The grabber board hardware is available and its firmware is currently under design.

## 4.2.2 Architecture Modules and Interfaces

### 4.2.2.1 Power

Due to the inherent redundancy of the architecture, no power protection countermeasures need to be taken. Each of the concentrator boards will be independently powered and will supply power to the daughterboards plugged to it. Thus cutting off power to one of the concentrator boards will not affect the system at all provided that we deploy a topology utilizing overlapping sensors attached to different concentrators. As more concentrators get powered off, the system performance will start degrading as soon as all sensors deployed in a region belong to concentrators that have been powered off.

Moreover, the grabber board is bus powered, thus its availability regarding power depends on the power availability of the embedded PC it is attached to.

#### 4.2.2.2 Security

Due to its nature, the system can be utilized as an anti-tampering mechanism by examining in real time the audio signals captured by sensors that are attached to its components. Moreover the communication between the concentrators and the grabber will be encrypted to guarantee data integrity. Code integrity is ensured by making the design non-upgradeable in the field. Finally, data freshness will be ensured by attaching a monotonically increasing label to each chunk of data captured.

#### 4.2.2.3 Status and Configuration

The node will inform upper layers regarding its status (detection of tampering attempt, sensors activated) and will allow them to configure the sensors to activate.

#### 4.2.3 Metrics

The node will provide metrics on its availability, tamper resilience and data freshness computed at real time as well as Boolean metrics describing its capabilities (such as ability to reconfigure the sensors utilized).

### 4.3 System of Embedded System, SoES

#### 4.3.1 Description

This section illustrates the technologies that will be employed to develop the power node prototype.

Leveraging the results gained on pSHIELD and considering the current evolution of nSHIELD, we began an in-depth analysis geared towards the identification of technologies that shall ease the composability of heterogeneous SHIELD nodes.

Distributed real-time and embedded (DRE) systems are a class of real-time systems formed through a composition of predominantly legacy, closed and statically scheduled real-time subsystems, which comprise over-provisioned resources to deal with worst-case failure scenarios. The formation of the system-of-systems leads to a new range of faults that manifest at different granularities for which no statically defined fault tolerance scheme applies.

The selected technologies will be used to endow SPD functionalities to legacy systems that, because of their nature, are not SHIELD compliant. In particular, we envision applying the aforementioned results to an Avionic System. Due to the fact that avionic systems are dominated by very stringent security policies, the integration of nSHIELD in such environment is extremely challenging and valuable.

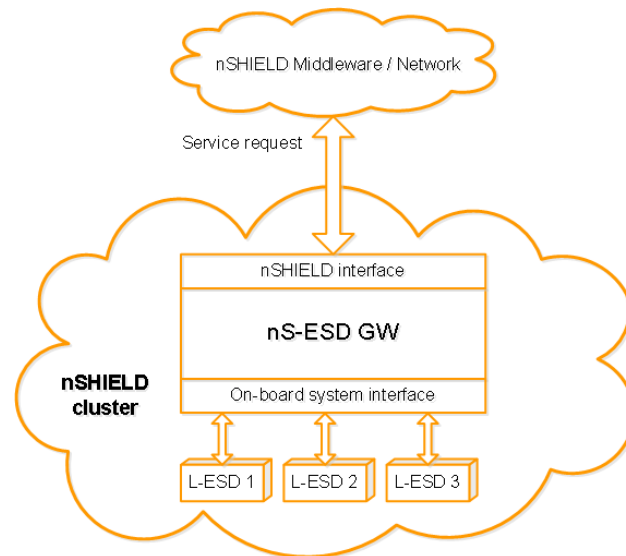
So far, a subset of technologies has been identified and some of them verified. Once the set of the technologies will be consolidated the integration process will be triggered. The result of such integration will be an IP that will expose the following features:

- Monitor interface;
- Data integrity;
- Encryption / Decryption;
- Dynamic Reconfiguration (complete and partial);
- Abnormal event detection;
- Diagnostic.

The process of technologies selection and technologies integration is tightly connected to the avionic application scenario; since the application scenario has not been consolidated yet, an adaptation of the exposed features is foreseen.



More specifically, a FPGA IP will be developed, according to the nSHIELD Overall Architecture as depicted in D2.3. It will represent an nS-ESD GW (nSHIELD Embedded System Device Gateway) that will provide enhanced capabilities in terms of security and dependability to the cluster where it will be integrated.



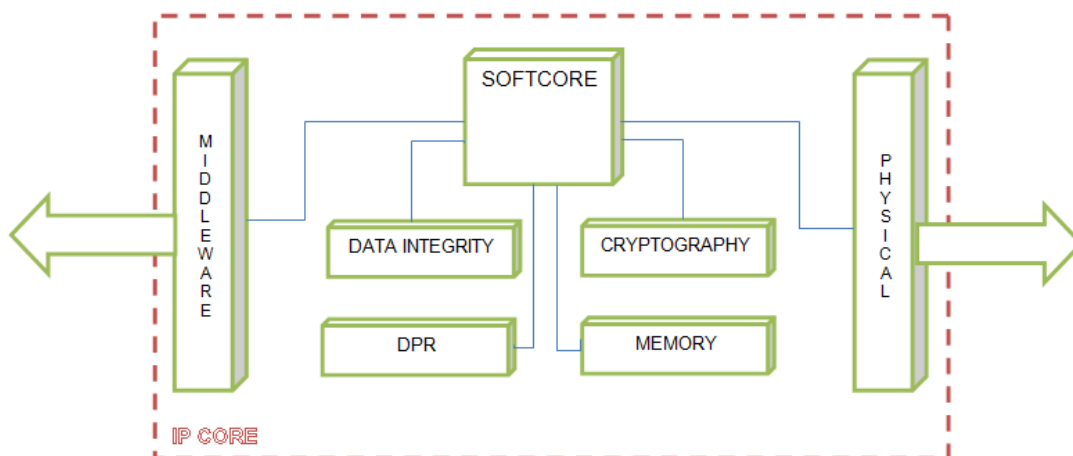
**Figure 4-6: nS-ESD GW Integration**

As shown on the picture above reported, the FPGA-based NS-ESD GW will operate as an nSHIELD adapter that interconnects the middleware and the node layers. To achieve the dependability and to assure performances, a Softcore microprocessor will be synthesized onto FPGA to support the application-specific customization. The implementation of the aforementioned NS-ESD GW implies the development of custom hardware controllers and software drivers, because of the heterogeneous nature of legacy nodes.

The overall NS-ESD GW design process will be performed in adherence to the RTCA/DO-254 and RTCA/DO-178B standard.

### 4.3.2 Architecture Modules and Interfaces

As reported by the Node Layer Logical View depicted in the D2.4, the FPGA-based IP Core will include different modules to implement specific capabilities.



**Figure 4-7: IP Logical Architecture.**



As shown in the picture above, the internal outlined architecture consists of a set of different components whose synergy will foster the integration of legacy embedded systems in the SHIELD architecture.

According to the logical view of the FPGA-based NS-ESD GW Core, the modules that will be developed are the following:

- Monitor module;
- Coordination module;
- Security module.

More specifically, in the context of this partitioning it's possible to identify elements that are involved to implement security, communication and monitoring functions.

Going further, a plethora of technological solutions is already made available for almost every single block highlighted in the previous figure. Which one to choose is a delicate balance among antagonistic aspects as costs, usefulness, technical complexities and design trade-offs

Having said this, a brief survey about every technological solution available for some blocks is briefly presented as follows:

- **DR / DPR AND DATA INTEGRITY:** manages the FPGA configuration process, increasing security and dependability levels of the node, transitioning the device among different operational modes. Depending on the speed of the reconfiguration process and on budget constraints: a wide variety of parallel and serial flash memory devices are supported. Newer devices offer dynamic reconfiguration port (DRP) as a mean of achieving partial dynamic reconfiguration features, highly valuable in the nSHIELD context. A high degree of design security can be attained thanks to the built-in AES decryption logic which decrypts on the fly the configuration bit stream. The chance to update system configuration bit streams on the field and remotely is provided as well with "multiboot" and "fallback" features.
- **SOFTCORE:** a wide range of embedded processing solutions: high performance, low power, and very low cost options are available (MicroBlaze, Leon3, NIOSII, ARM).
- **CRYPTOGRAPHY:** Cryptography plays a vital role for securing information exchange. Cryptographic algorithms impose tremendous processing power demands that can be a bottleneck in high-speed and real time networks. The implementation of a cryptographic algorithm must achieve high processing rate to fully utilize the available network bandwidth. To follow the variety and the rapid changes in algorithms and standards, a cryptographic implementation must also support different algorithms and should be upgradeable in field. For all these reasons, a reconfigurable/upgradable FPGA based HW accelerator could provide software-like flexibility with hardware-like performances

#### **4.3.2.1 Monitor Module**

In order to foster the integration and communications between legacy power Node and SHIELD components (Middleware, Nodes, etc.), two FPGA controllers will be developed. In the logical view above these controllers are identified as middleware and physical. Through the middleware controller the NS-ESD GW Core converts a service request from the nSHIELD component to a logical format that legacy systems in the lower layer can understand. This component can be seen as an interface to the SHIELD architecture. Likewise, the physical controller will be used to supply proxy services for nodes with a non-standard physical communication. According to this, it can be seen as an interface to the legacy embedded systems.

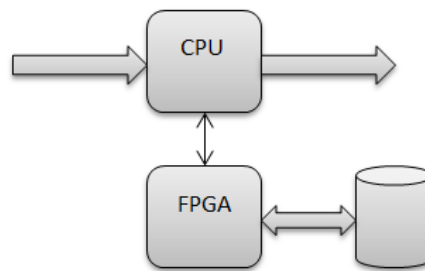
Middleware and physical controllers will provide configuration and status services through both hardware and software.

#### 4.3.2.2 Coordination Module

The presence of a softcore provides coordination capabilities. In the context of an FPGA-based solution it is possible to implement different microprocessors using logic synthesis. Different softcores can be instantiated on the FPGA according to the specific technology that will be identified to be used for the demonstrator and to specific features required. Some of softcores that can be used are:

- Microblaze: 32bit RISC-based proprietary soft-processor by Xilinx;
- Nios II: 32bit RISC proprietary softcore by Altera;
- Leon3/4: 32bit open source soft-processor based on SPARC V8 RISC architecture by Aeroflex Gaisler;
- Cortex M1: 32bit RISC ARM proprietary processor licensed by ARM Holdings;
- OpenSPARC T1: 64bit open source softcore RISC architecture by Oracle.

Thanks to the softcore it will be possible to manage the resources allocation and to balance services with the aim to ensure the system reliability with a dynamic load distribution. The processor will be exploited to execute balancing algorithms, safety algorithms, etc. The development of hardware accelerators (HA) is foreseen. An HA is a hardware implementation of a specific operation or function e.g. cryptography. There are several algorithms and functions that are specially tailored to be executed by an HA rather than a microprocessor due to their parallel nature. Using the HA is possible to optimize the system performance in a way that is not possible with traditional off-the-shelf processor.



**Figure 4-8: Hardware Accelerator**

#### 4.3.2.3 Security module

Several components will be developed as parts of the security module.

The component identified as DPR will be in charge of managing the FPGA configuration. The purpose of this functional block is to increase security and dependability aspects of the node allowing the device to change between different operational modes. Thanks to this component it will also be possible to run multiple applications on the same device enhancing the dependability and redundancy.

A Data Integrity controller will be included into the NS-ESD GW architecture to process data by implementing cyclic redundancy check. This component validates the content of each data packet by calculating a new CRC value and matching it to the one contained in the packet. The core also checks the ordering of the received packets by monitoring the contained sequence number in each packet.

It's also scheduled the implementation of a Secure Storage controller to protect the confidentiality and integrity of processed data. In conjunction with an Encrypt/Decrypt controller it will be possible to ensure long term storage of sensitive data improving security aspects of the node.

#### 4.3.3 Metrics

The nS-ESD GW Core will provide metrics as defined in D2.5 (Preliminary SPD Metric Specifications). In particular, with the aim to ensure a fulfil integration of such IP with SHIELD architecture, security, dependability and privacy metrics will be implemented.

Digital signature policies and data freshness metrics will be implemented to confer Security aspects on the FPGA-based solution. These metrics will be implemented in the Security Module.

The runtime reconfiguration metric will be implemented to improve the Dependability of the node during both normal operation and fault conditions. This metric will be used in the DPR module.

The secure key distribution and the storage of private information will be provided to give an appropriate level of Privacy to the node. These metrics will be implemented as core of the Encrypt/Decrypt and Data Integrity controllers.

Being the node an nS-ESD GW, also the node availability metric will be implemented to provide the composability of the IP inside a legacy cluster.

## **4.4 GPU accelerated hashing and hash lookup mechanism**

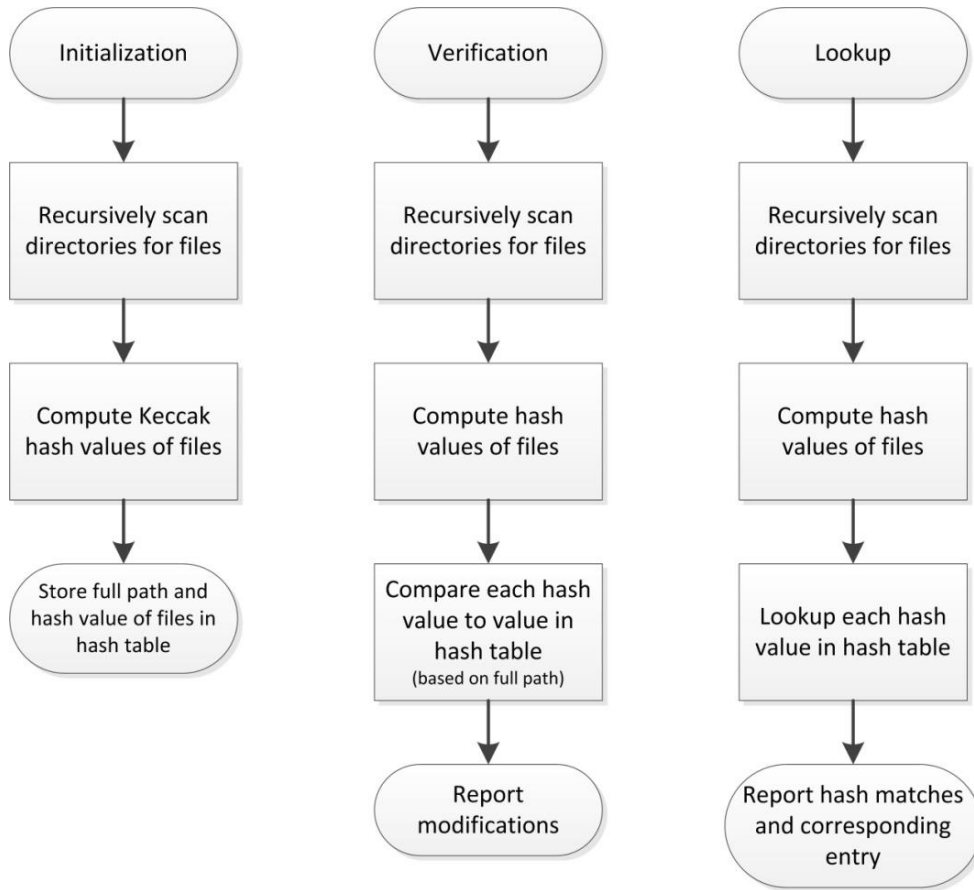
### **4.4.1 General description**

This implementation involves the development of a lightweight, efficient, GPU accelerated hashing and hash lookup mechanism utilizing the CUDA GPGPU toolkit and exploiting the highly parallel nature of modern GPUs.

Main operations:

- Compute a digest
- Compare two digests
- Insert a new digest in the hash table
- Search for a digest in the hash table
- Delete or Move detected malicious file (optional)

A sketch of the basic functionality of the utility can be seen on the following figure.



**Figure 4-9: Functionality sketch of GPU-accelerated hashing mechanism**

The mechanism, as implemented in its basic form, is a simple file integrity checker. It involves a command line interface where at the time of execution a directory has to be provided. The folder and subfolders are accessed recursively and hash values are computed for all the included files. The mechanism utilizes the recently selected SHA-3 hash function (i.e. the Keccak hash function [28]). The hash values are kept in a table structure with two fields: full path & hash value.

Once this first initialization phase is over, the utility waits for another folder to scan. Again, it recursively scans the folder and looks for an entry that matches each file’s full path. If there is no such entry the file is considered unknown/new and a message is displayed. If it does find the same full path entry but the hash values do not match, another message is displayed. In this form, the tool can be used for pre- or post-installation audits, integrity checks on firmware files stored in secure locations etc. If the full path/filename and hash value are the same, no message is displayed. This basic functionality can be seen in the Figure 4-10, where file “example.txt” did not exist during initialization and file “scanner.c” has been modified since the initialization phase (i.e. hash values do not match).

```

al ex@al ex- desktp: ~/KeccakReferenceAndOptimized/bin$ ./KeccakReference test
Table created.
Enter directory to scan :
test
FILE /home/al ex/KeccakReferenceAndOptimized/bin/test/example.txt is new
FILE /home/al ex/KeccakReferenceAndOptimized/bin/test/scanner.c has been modified.
Total time = 0.210359 seconds
al ex@al ex- desktp: ~/KeccakReferenceAndOptimized/bin$ █
  
```

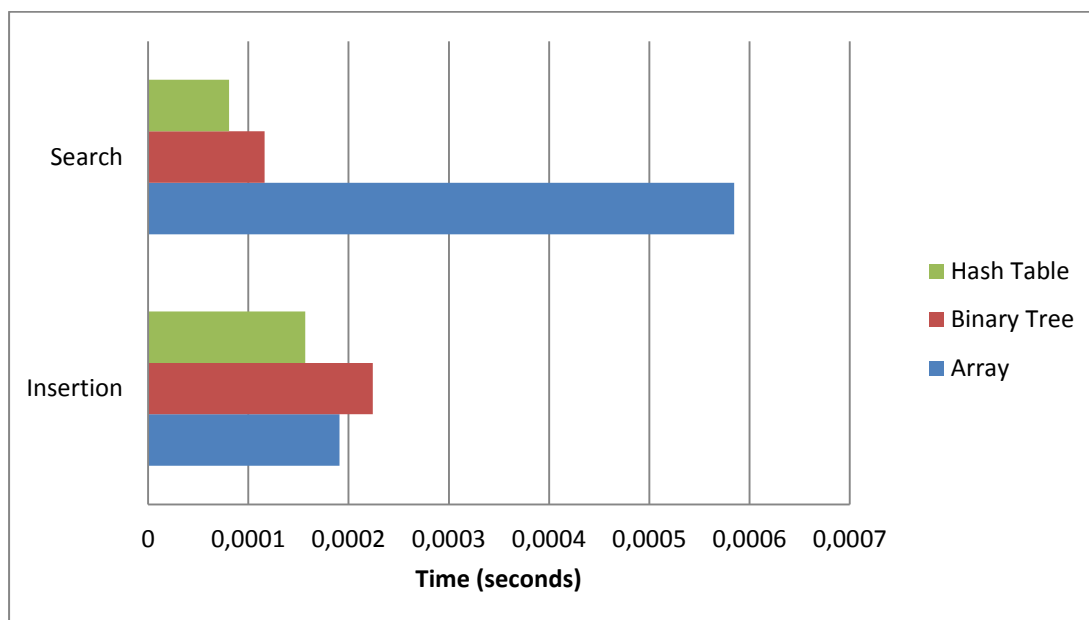
**Figure 4-10: Basic file check utilizing the SHA-3 hash function**

In its hash-matching form, the utility does not look up values in the hash table based on full path. Instead, it scans the whole hash table looking for matches for the hash it just calculated. This way, it can report back if the specific file exists in the initial hash table and what that file was. Such a mechanism could be useful for malware detection on local disks or network monitoring (if, e.g., during initialization, known malware are used as input).

#### 4.4.2 Development & performance comparisons

An initial CUDA implementation of the Keccak hashing mechanism was compared to the reference serial implementation as well as an optimized serial implementation, both of which are available on the official website [29].

Before proceeding with optimizing the GPU implementation, the data structure to be used had to be finalized. To that end, the performance of alternative structures was examined and compared. In specific, array, binary tree and hash table structures were investigated, focusing on the insertion time (i.e. how long it takes to insert a new entry into the data structure during initialization) and search time (i.e. how long it takes to lookup a specific entry in the data structure). The results of said comparison can be seen in Figure 4-11, where it is evident that the hash table has an advantage over alternative structures.



**Figure 4-11: Data structure performance comparison for CUDA-accelerated implementation Dependable self-x Technologies**

Having finalized the data structure to be used, development could focus on the optimization of the CUDA hashing mechanism. Some preliminary results appear below. As is evident from the graph, the GPU-accelerated version of the hashing mechanism has a significant advantage in terms of the actual hashing performance, but also suffers extra I/O overhead because of the data transfer from host to GPU. The CUDA implementation will be further optimized to allow for greater speed-up times.

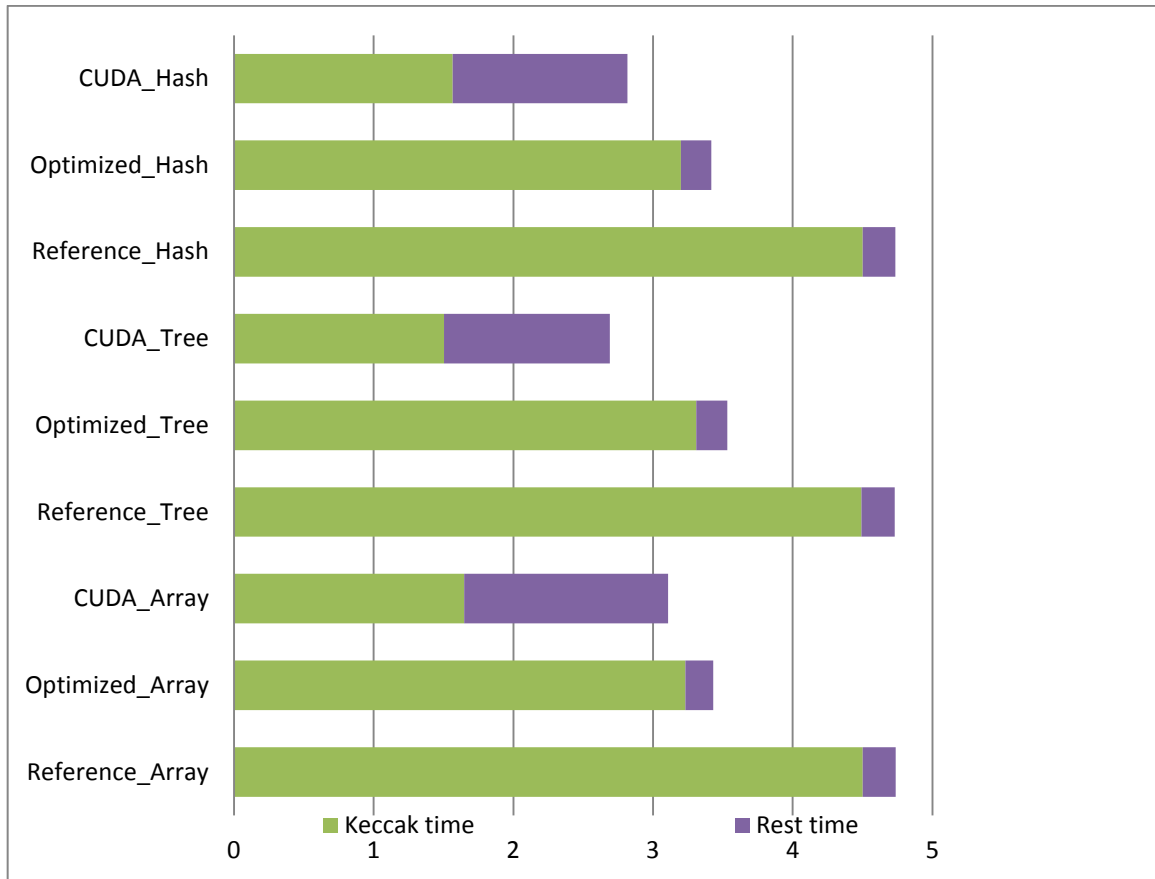


Figure 4-12: Comparison of execution and I/O time of SHA-3 implementations

## 5 Dependable self-x Technologies

The main research areas targeted by the dependable self-x technologies are the following:

- Selection of the appropriate platform to be used as a basis for development.
- An anonymity service targeting applications with demanding privacy needs.
- Mechanisms for DDoS attack mitigation.

### 5.1 Platform Selection

According to the task requirements the research was focused to define a prototype of a scalable node family in order to cover the three node typologies and to be useful for the four application scenario.

The chosen platform integrates in a very small (18X68 mm) and low power board three main devices with different and orthogonal features: micro controller, FPGA and DSP.

The scalability can be achieved deciding which components integrate in the final product or the size of the component itself (for instance the dimension of the FPGA or the size of the microprocessor memory).

The general schematic of the prototype is shown below (source: OMBRAv2 HW MANUAL- Issue 9).

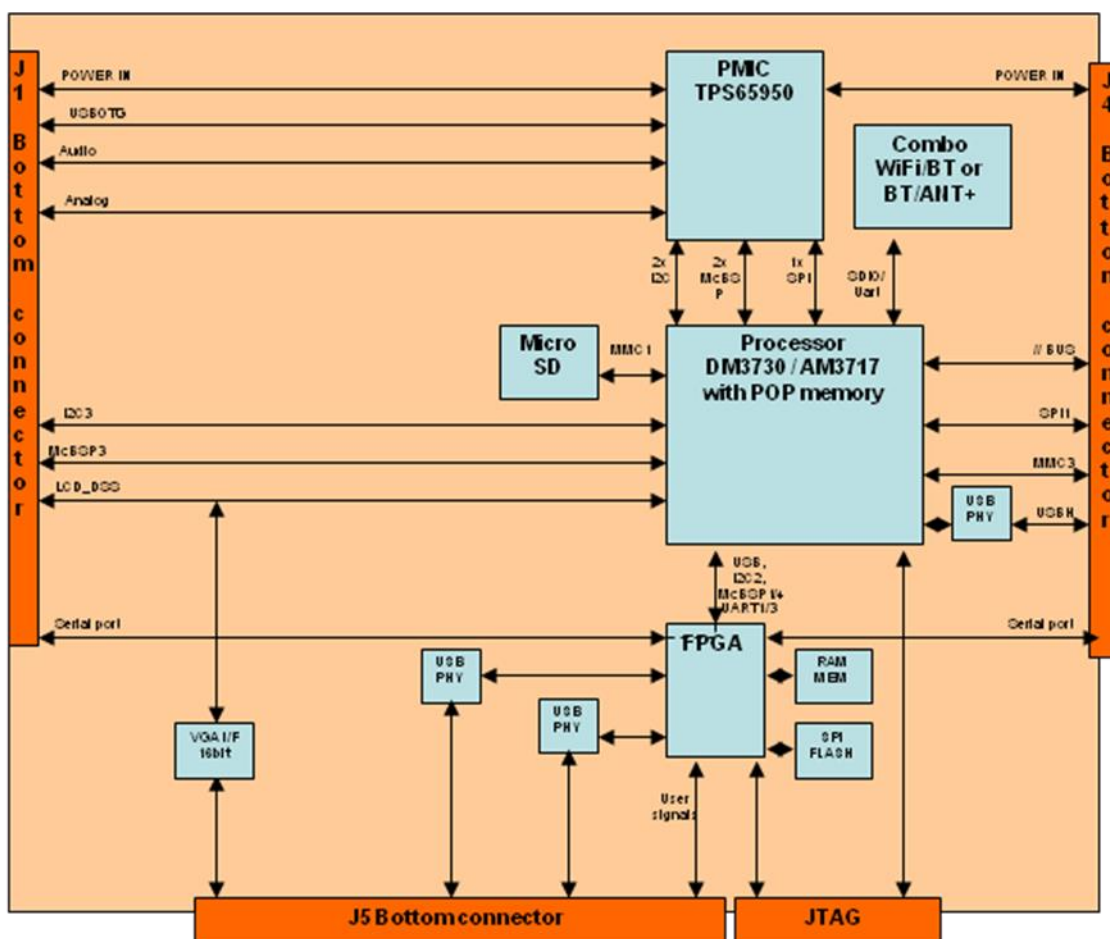


Figure 5-1: Prototype schematic

The following table summarize the prototype features (source: OMBRAv2 HW MANUAL- Issue 9).

Table 5-1: Prototype features

<b>Description</b>	<b>Full Version</b>	<b>Standard Version</b>	<b>Lite Version</b>
<i>Processor</i>	Ti DM3730	Ti DM3730	Ti AM3717
<i>ARM CortexA8 processor speed</i>	1Ghz	1Ghz	1Ghz
<i>DSP</i>	TMS320C64+	TMS320C64+	-
<i>DSP processor speed</i>	800Mhz	800Mhz	-
<i>Ram POP Memory (LpDDR)</i>	1024 Mbyte	512 Mbyte	512 Mbyte
<i>Flash POP Memory (Nand)</i>	-	512 Mbyte	-
<i>Flash Disk</i>	On uSD card up to 32GB	On uSD card up to 32GB	On uSD card up to 32GB
<i>Video 3D acceleration</i>	PowerVR SGX530	PowerVR SGX530	PowerVR SGX530
<i>Wireless connection WiFi</i>	IEEE 802.11 b/g/n (option (*))		
<i>Wireless connection Bluetooth</i>	2.1+ EDR class 1.5	2.1+ EDR class 1.5	2.1+ EDR class 1.5
<i>Wireless connection PAN</i>	ANT+ (option (*))	ANT+ (option (*))	ANT+ (option (*))
<i>Wireless Bluetooth/WiFi/ANT+ antenna</i>	Unified u.FI connector	Unified u.FI connector	Unified u.FI connector
<i>Expansions connector</i>	J1, J4, J5	J1, J4	J1, J4
<i>USB 2.0 OTG</i>	1 connected to ARM (HS,FS,LS)	1 connected to ARM (HS,FS,LS)	1 connected to ARM (HS,FS,LS)
<i>USB 2.0 HOST</i>	1 connected to ARM (HS only)	1 connected to ARM (HS only)	1 connected to ARM (HS only)
<i>USB 1.1 HOST</i>	connected to FPGA (FS only)	connected to FPGA (FS only)	connected to FPGA (FS only)
<i>USB 1.1 HOST/DEVICE sw configurable</i>	connected to FPGA (FS, LS)	connected to FPGA (FS, LS)	connected to FPGA (FS,LS)
<i>Audio Stereo</i>	Mic In, Line-in and Line-out	Line-in and Line-out	Line-in and Line-out
<i>Video VGA</i>	16 bit (VGA and XGA)	16 bit (VGA and XGA)	16 bit (VGA and XGA)
<i>Video Digital Output connection</i>	DSS 24 bit 1v8 levels TV composite/S-VIDEO	DSS 24 bit 1v8 levels	DSS 24 bit 1v8 levels

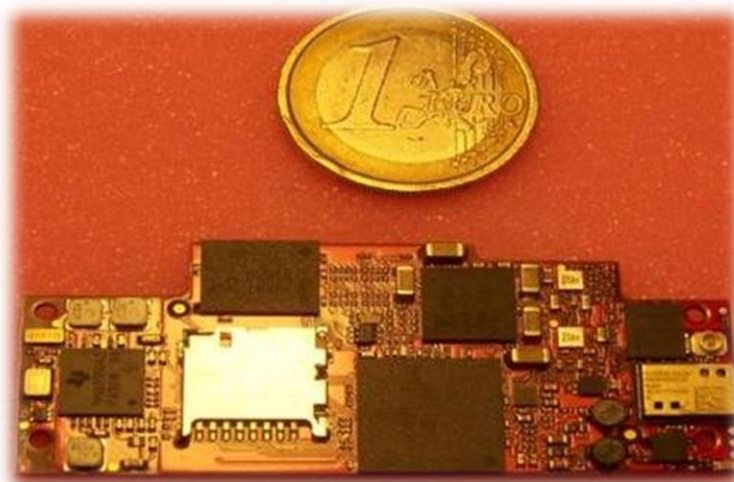


<i>Video Digital Input connection</i>	Omap 12 bit Camera interface		
<i>FPGA</i>	Xilinx Spartan6 (XC6SLX16-2CPG196I)	Xilinx Spartan6 (XC6SLX16-2CPG196I)	Xilinx Spartan6 (XC6SLX16-2CPG196I)
<i>FPGA connections between FPA and OMAP</i>	1xUSB-FS, McBSP1, McBSP4, 2xGPIO, Uart3, (2Wires), I2C-2, Uart1 (4wires), 2xGPIO	1xUSB-FS, McBSP1, McBSP4, Uart1 (4wires), Uart3/4(2Wires), 2xGPIO, I2C-2, SPI1, 6xGPIO	1xUSB-FS, McBSP1, McBSP4, Uart1 (4wires), Uart3/4(2Wires), 2xGPIO, I2C-2, SPI1, 6xGPIO
<i>FPGA pins available for the user</i>	4+2 on J1/J4 (1v8 IO-level) 27 on J5 (1v8 IO-level)	4 on J1/J4 (1v8 IO-level) 15 on J1/J4 (1v8 IO-level)	4 on J1/J4 (1v8 IO-level) 15 on J1/J4 (1v8 IO-level)
<i>Debug connection</i>	JTAG both for processor and FPGA	JTAG both for processor and FPGA	JTAG both for processor and FPGA
<i>Power Supply Range</i>	3.2..4.2 Vcc, 2.5W	3.2..4.2 Vcc, 2.5W	3.2..4.2 Vcc, 2W
<i>Temperature range (components)</i>	-40..+55 C	-40..+55 C	-20..+55 C

The current prototype just includes three possible levels of scalability but it is possible to build any descendent platform starting from the Full version.

The main component is the Processor and others component can be used as co-processor or as new features-hardware.

The figure below shows the picture of the OMBRA board with respect to 1 euro coin.



**Figure 5-2: The OMBRA board**

Hereafter the main features of the three main components of the OMBRA Board taken from the component datasheets itself: Processor, FPGA and DSP.

General feature of the chosen processor (DM3730/25 Digital Media Processors):

- **DM3730/25 Digital Media Processors:**
  - Compatible with OMAP™ 3 Architecture
  - ARM® Microprocessor (MPU) Subsystem
    - Up to 1-GHz ARM® Cortex™-A8 Core  
Also supports 300, 600, and 800-MHz operation
    - NEON™ SIMD Coprocessor
  - High Performance Image, Video, Audio (IVA2.2™) Accelerator Subsystem
    - Up to 800-MHz TMS320C64x+™ DSP Core  
Also supports 260, 520, and 660-MHz operation
    - Enhanced Direct Memory Access (EDMA) Controller (128 Independent Channels)
    - Video Hardware Accelerators
  - POWERVR SGX™ Graphics Accelerator (DM3730 only)
    - Tile Based Architecture Delivering up to 20 MPoly/sec
    - Universal Scalable Shader Engine: Multi-threaded Engine Incorporating Pixel and Vertex Shader Functionality
    - Industry Standard API Support: OpenGL ES 1.1 and 2.0, OpenVG1.0
    - Fine Grained Task Switching, Load Balancing, and Power Management
    - Programmable High Quality Image Anti-Aliasing
  - Advanced Very-Long-Instruction-Word (VLIW) TMS320C64x+™ DSP Core
    - Eight Highly Independent Functional Units
    - Six ALUs (32-/40-Bit); Each Supports Single 32-bit, Dual 16-bit, or Quad 8-bit, Arithmetic per Clock Cycle
    - Two Multipliers Support Four 16 x 16-Bit Multiplies (32-Bit Results) per Clock Cycle or Eight 8 x 8-Bit Multiplies (16-Bit Results) per Clock Cycle
- Load-Store Architecture With Non-Aligned Support
- 64 32-Bit General-Purpose Registers
- Instruction Packing Reduces Code Size
- All Instructions Conditional
- Additional C64x+™ Enhancements
  - Protected Mode Operation
  - Expectations Support for Error Detection and Program Redirection
  - Hardware Support for Modulo Loop Operation
- C64x+™ L1/L2 Memory Architecture
  - 32K-Byte L1P Program RAM/Cache (Direct Mapped)
  - 80K-Byte L1D Data RAM/Cache (2-Way Set-Associative)
  - 64K-Byte L2 Unified Mapped RAM/Cache (4-Way Set-Associative)
  - 32K-Byte L2 Shared SRAM and 16K-Byte L2 ROM
- C64x+™ Instruction Set Features
  - Byte-Addressable (8-/16-/32-/64-Bit Data)
  - 8-Bit Overflow Protection
  - Bit-Field Extract, Set, Clear
  - Normalization, Saturation, Bit-Counting
  - Compact 16-Bit Instructions
  - Additional Instructions to Support Complex Multiplies
- External Memory Interfaces:
  - SDRAM Controller (SDRC)
    - 16, 32-bit Memory Controller With 1G-Byte Total Address Space
    - Interfaces to Low-Power SDRAM
    - SDRAM Memory Scheduler (SMS) and Rotation Engine
  - General Purpose Memory Controller (GPMC)
    - 16-bit Wide Multiplexed Address/Data

The double boot mode, through the micro SD, allows to use the processor as a Linux or Windows CE embedded PC, with the possibility to work with the Windows or the GNU tool chain.

The selected FPGA is a Xilinx Spartan 6 – 16. All Xilinx tool chain can be used for developing HW and/or SW devices (for example a MicroBlaze was mapped on the FPGA).

## Summary of Virtex-6 FPGA Features

- Three sub-families:
  - Virtex-6 LXT FPGAs: High-performance logic with advanced serial connectivity
  - Virtex-6 SXT FPGAs: Highest signal processing capability with advanced serial connectivity
  - Virtex-6 HXT FPGAs: Highest bandwidth serial connectivity
- Compatibility across sub-families
  - LXT and SXT devices are footprint compatible in the same package
- Advanced, high-performance FPGA Logic
  - Real 6-input look-up table (LUT) technology
  - Dual LUT5 (5-input LUT) option
  - LUT/dual flip-flop pair for applications requiring rich register mix
  - Improved routing efficiency
  - 64-bit (or two 32-bit) distributed LUT RAM option per 6-input LUT
  - SRL32/dual SRL16 with registered outputs option
- Powerful mixed-mode clock managers (MMCM)
  - MMCM blocks provide zero-delay buffering, frequency synthesis, clock-phase shifting, input-jitter filtering, and phase-matched clock division
- 36-Kb block RAM/FIFOs
  - Dual-port RAM blocks
  - Programmable
    - Dual-port widths up to 36 bits
    - Simple dual-port widths up to 72 bits
  - Enhanced programmable FIFO logic
  - Built-in optional error-correction circuitry
  - Optionally use each block as two independent 18 Kb blocks
- High-performance parallel SelectIO™ technology
  - 1.2 to 2.5V I/O operation
  - Source-synchronous interfacing using ChipSync™ technology
  - Digitally controlled impedance (DCI) active termination
  - Flexible fine-grained I/O banking
  - High-speed memory interface support with integrated write-leveling capability
- Advanced DSP48E1 slices
  - 25 x 18, two's complement multiplier/accumulator
  - Optional pipelining
  - New optional pre-adder to assist filtering applications
  - Optional bitwise logic functionality
  - Dedicated cascade connections
- Flexible configuration options
  - SPI and Parallel Flash interface
  - Multi-bitstream support with dedicated fallback reconfiguration logic
  - Automatic bus width detection
- System Monitor capability on all devices
  - On-chip/off-chip thermal and supply voltage monitoring
  - JTAG access to all monitored quantities
- Integrated interface blocks for PCI Express® designs
  - Compliant to the PCI Express Base Specification 2.0
  - Gen1 (2.5 Gb/s) and Gen2 (5 Gb/s) support with GTX transceivers
  - Endpoint and Root Port capable
  - x1, x2, x4, or x8 lane support per block
- GTX transceivers: up to 6.6 Gb/s
  - Data rates below 480 Mb/s supported by oversampling in FPGA logic.
- GTH transceivers: 2.488 Gb/s to beyond 11 Gb/s
- Integrated 10/100/1000 Mb/s Ethernet MAC block
  - Supports 1000BASE-X PCS/PMA and SGMII using GTX transceivers
  - Supports MII, GMII, and RGMII using SelectIO technology resources
  - 2500Mb/s support available
- 40 nm copper CMOS process technology
- 1.0V core voltage (-1, -2, -3 speed grades only)
- Lower-power 0.9V core voltage option (-1L speed grade only)
- High signal-integrity flip-chip packaging available in standard or Pb-free package options

TMS320C64 outline:

- **High-Performance Fixed-Point Digital Signal Processor (TMS320C6413/C6410)**
  - TMS320C6413
    - 2-ns Instruction Cycle Time
    - 500-MHz Clock Rate
    - 4000 MIPS
  - TMS320C6410
    - 2.5-ns Instruction Cycle Time
    - 400-MHz Clock Rate
    - 3200 MIPS
  - Eight 32-Bit Instructions/Cycle
  - Fully Software-Compatible With C64x™
  - Extended Temperature Devices Available
- **VelociTI.2™ Extensions to VelociTI™ Advanced Very-Long-Instruction-Word (VLIW) TMS320C64x™ DSP Core**
  - Eight Highly Independent Functional Units With VelociTI.2™ Extensions:
    - Six ALUs (32-/40-Bit), Each Supports Single 32-Bit, Dual 16-Bit, or Quad 8-Bit Arithmetic per Clock Cycle
    - Two Multipliers Support Four 16 x 16-Bit Multiplies (32-Bit Results) per Clock Cycle or Eight 8 x 8-Bit Multiplies (16-Bit Results) per Clock Cycle
  - Load-Store Architecture With Non-Aligned Support
  - 64 32-Bit General-Purpose Registers
  - Instruction Packing Reduces Code Size
  - All Instructions Conditional
- **Instruction Set Features**
  - Byte-Addressable (8-/16-/32-/64-Bit Data)
  - 8-Bit Overflow Protection
  - Bit-Field Extract, Set, Clear
  - Normalization, Saturation, Bit-Counting
  - VelociTI.2™ Increased Orthogonality
- **VelociTI.2™ Extensions to VelociTI™ Advanced Very-Long-Instruction-Word (VLIW) TMS320C64x™ DSP Core**
- **L1/L2 Memory Architecture**
  - 128K-Bit (16K-Byte) L1P Program Cache (Direct Mapped)
  - 128K-Bit (16K-Byte) L1D Data Cache (2-Way Set-Associative)
  - 2M-Bit (256K-Byte) L2 Unified Mapped RAM/Cache [C6413] (Flexible RAM/Cache Allocation)
  - 1M-Bit (128K-Byte) L2 Unified Mapped RAM/Cache [C6410] (Flexible RAM/Cache Allocation)
- **Endianess: Little Endian, Big Endian**
- **32-Bit External Memory Interface (EMIF)**
  - Glueless Interface to Asynchronous Memories (SRAM and EPROM) and Synchronous Memories (SDRAM, SBSRAM, ZBT SRAM, and FIFO)
  - 512M-Byte Total Addressable External Memory Space
- **Enhanced Direct-Memory-Access (EDMA) Controller (64 Independent Channels)**
- **Host-Port Interface (HPI) [32-/16-Bit]**
- **Two Multichannel Audio Serial Ports (McASPs) - with Six Serial Data Pins each**
- **Two Inter-Integrated Circuit (I<sup>2</sup>C) Buses - Additional GPIO Capability**
- **Two Multichannel Buffered Serial Ports**
- **Three 32-Bit General-Purpose Timers**
- **Sixteen General-Purpose I/O (GPIO) Pins**
- **Flexible PLL Clock Generator**
- **On-Chip Fundamental Oscillator**
- **IEEE-1149.1 (JTAG†) Boundary-Scan-Compatible**
- **288-Pin Ball Grid Array (BGA) Packages (GTS and ZTS Suffixes), 1.0-mm Ball Pitch**
- **0.13-μm/6-Level Cu Metal Process (CMOS)**
- **3.3-V I/Os, 1.2-V Internal**

Users and designers that want to implement self-x technologies can use the powerful reconfigurability features of the Ombra board.

For instance it is possible to change hardware functionalities reprogramming the FPGA and chaining on the fly hardware parameters or algorithms: it is possible to think that micro-controller application can reprogram the FPGA or a lot subset. It is also possible to use the FPGA IOs in order to configure different interfaces, for instance for different sensor to use in several application, maintaining the same kernel.

One of the most interesting methods for security implementation is cryptography. Several algorithms were studied and implemented; the most important point for such task is the trade-off between security level and computational complexity in order of calculation time and hardware resources.

Speaking about embedded system this kind of problem is more important with respect to the reduced size in term of area and hardware resources of the target system.



The Ombra multi-core system allows a Hardware-Software co design in a very small area, this allows to build and run algorithms as the elliptic curves cryptography in a very short time.

The implemented demo shows how it is possible to run such algorithm in a very short time.

### **5.1.1 Metrics**

The purpose of this section is to correlate the prototype features with the metrics for the node level described in the D2.5 deliverable.

Generally speaking, the prototype implements a multi core platform able to meet the requirements of self-reconfigurability and self-adaptation both software and hardware levels. Self-reconfigurability and self-adaptation are the main features in order to implement SPD capabilities also at network and middleware level. In fact, the purpose of this prototype is to provide a complete and powerful platform for not only for the three node typology but also for the upper nSHIELD levels.

Note that several metrics described in D2.5 are not directly applicable in the hardware platform but their application in the upper levels can be facilitated by an appropriate hardware platform.

#### **5.1.1.1 Security**

Metrics related to cryptography are met as demonstrated with the demo (next section). The Montgomery's algorithm, that performs the point multiplication on an elliptic curve, is implemented in the prototype. The run speed allows using different bit lengths for the curve, allowing different security level. The possibility to use a reconfigurable hardware device could allow the application to change the cryptography parameter run time.

#### **5.1.1.2 Dependability**

The metrics of redundancy and reconfiguration can be easily achieved by the multi core platform. Redundancy is granted by the multi core and reconfiguration is granted by the reconfigurable hardware.

#### **5.1.1.3 Privacy**

The characteristics of the embedded platform ensure the possibility to store sensible and private data in non-accessible memory space. In addition an area of the FPGA (subsets of the block RAM for instance) can be allocated for this use.

#### **5.1.1.4 Composability**

The high flexibility of the node platform allows performing the listed dynamic and flexibility features.

#### **5.1.1.5 Performance**

Lightweight embedded operating system: the prototype micro-host can support Linux and windows CE.

#### **5.1.1.6 Interfaces**

The on board FPGA can be configured to directly implement custom devices or to implement connection interfaces for new devices.

### **5.1.2 Demo**

This chapter describes the implementation on the OMBRA platform of the Montgomery's algorithm that performs the point multiplication on an elliptic curve developed in work package 3.5, refer to section 6 for the technical details.

The target is to show the running algorithm on the implemented embedded Linux platform (Linux omap 2.6.32) and calculate the working time. It was needful to compile for the above mentioned Linux version the gnu gmp libraries.



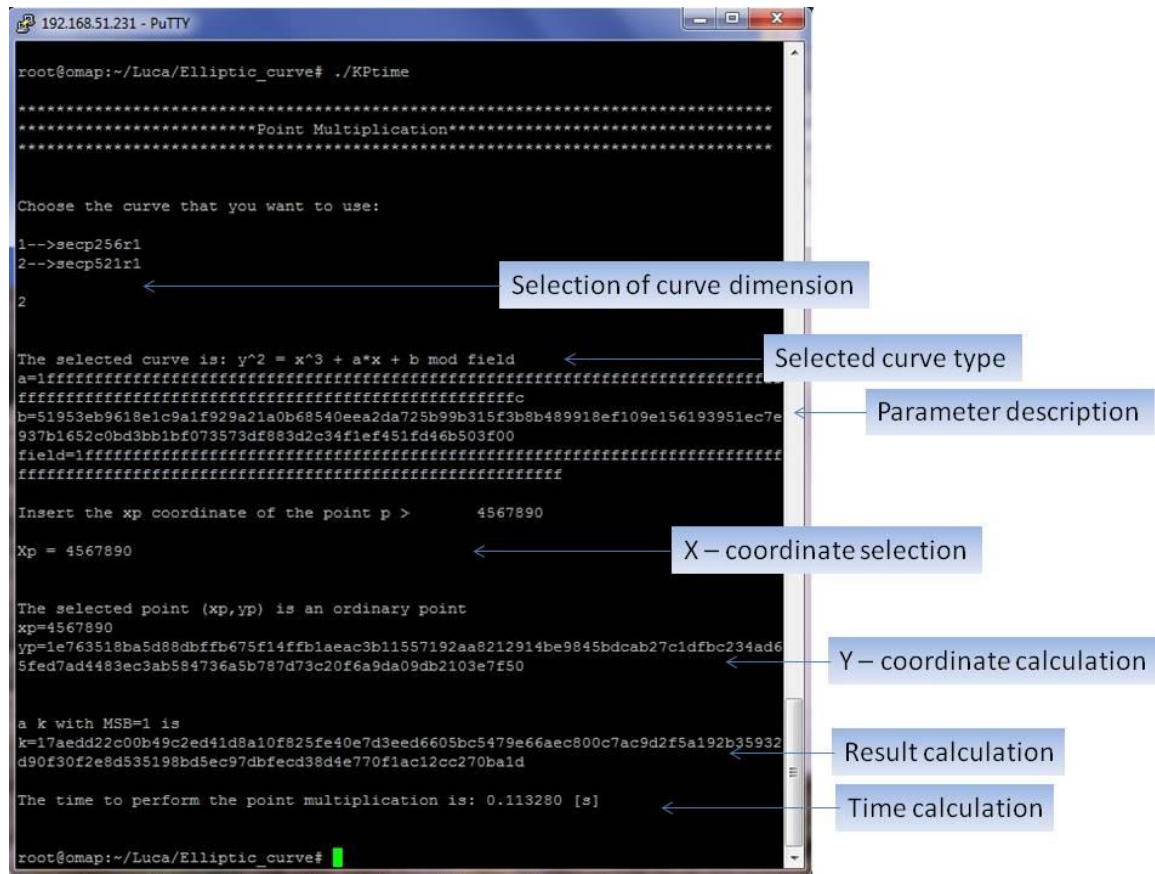


Figure 5-4: Demo for 521 bit curve

We perform the same experiments on an intel core i7 @3Ghz to compare the results. In the 256 bit length case the time to perform the point multiplication (change coordinates included) is 0.003895 seconds. In the 521 bit length case the time to perform the point multiplication (change coordinates included) is 0.017379 seconds. We can note that the computational times on the OMBRA platform are very good (ratio is around 6 in the 256 bit length and around 6,5 in the 521 bit length) considering that its processor runs at 1Ghz.

## 5.2 Anonymity Service

An anonymizer component is being developed for nSHIELD applications where personal location privacy must be preserved while enabling the system to provide location monitoring services. Such a service is essential in applications involving wearable/personal nodes where, on one hand, privacy concerns may arise from disclosing the service users' exact location while, on the other hand, users' location is essential to provide certain personalised services.

After an extensive state of the art review, the TinyCasper [30] scheme was selected and will be implemented, aiming to preserve personal location privacy via the well-established K-anonymity privacy concept, while enabling the system to provide location monitoring services. The implemented scheme will offer a resource-aware algorithm for applications where it is essential to minimize communication and computational cost. A quality-aware algorithm which minimizes the size of cloaked areas in order to generate more accurate aggregate locations may be developed at a later stage. Provisions will be made and original work will be extended in order to better match nSHIELD's architecture and hardware and to facilitate scenarios where the nodes are mobile. On the server side a simple graphical interface will be used in order to setup the various parameters and monitor the system.

### 5.2.1 Network Topology & Connectivity

As already mentioned, the system implemented relies on the K-anonymity privacy concept that aims to make a user indistinguishable from "K" of her neighbours.

In more detail, every sensor node is connected to its neighbours. Sensors of a node may have different sensing areas due to hardware limitations: for example RED sensors RS={J} have 12x12 sensing area, PURPLE sensors PS={H,I,B,F,E,D} have 5x5, and GREEN sensors GS={A,C,G,I} have 7x7. This is depicted in the figure below. Classes of sensors and their respective range must be set during initialization.

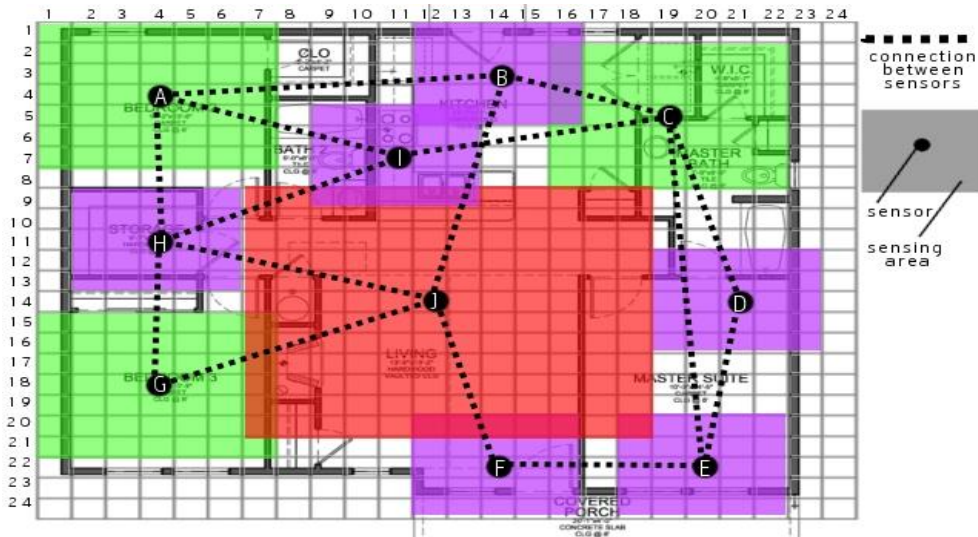


Figure 5-5: Overlay of anonymity service node topology over floor layout.

The location of every sensor and its sensing area is modelled as coordinates of a grid. The server of the system implements this grid and also keeps the accurate location of every connected node and its neighbouring nodes. The server is not aware of the type each sensor nor its sensing area as shown below.

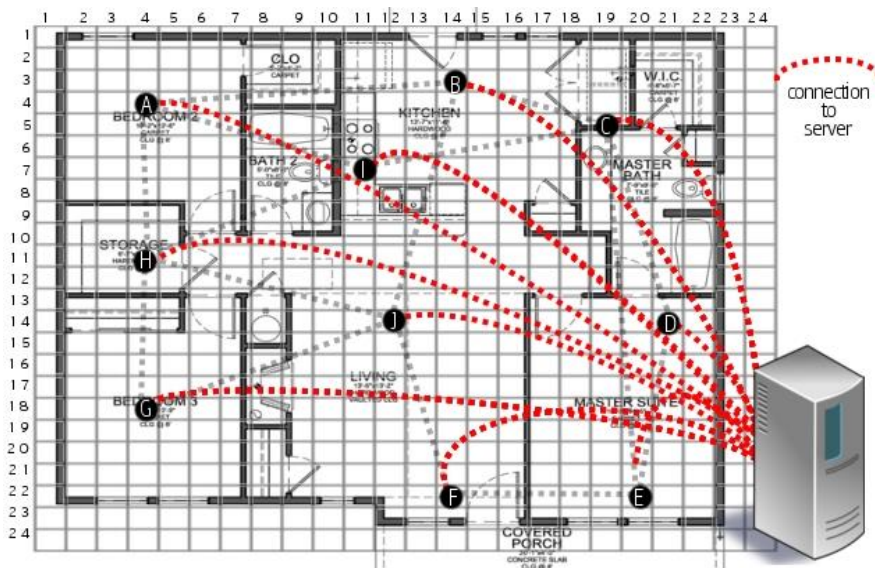


Figure 5-6: The anonymity server and its connection to nodes.



If a sensor moves to a new position and its sensing area is out of the grid limits, its type is downgraded to a lower one. The connectivity between sensors is also set by default until the sensor moves to another location as shown below. If a sensor moves, it informs the server about the new position and the server replies with the new neighbouring nodes.

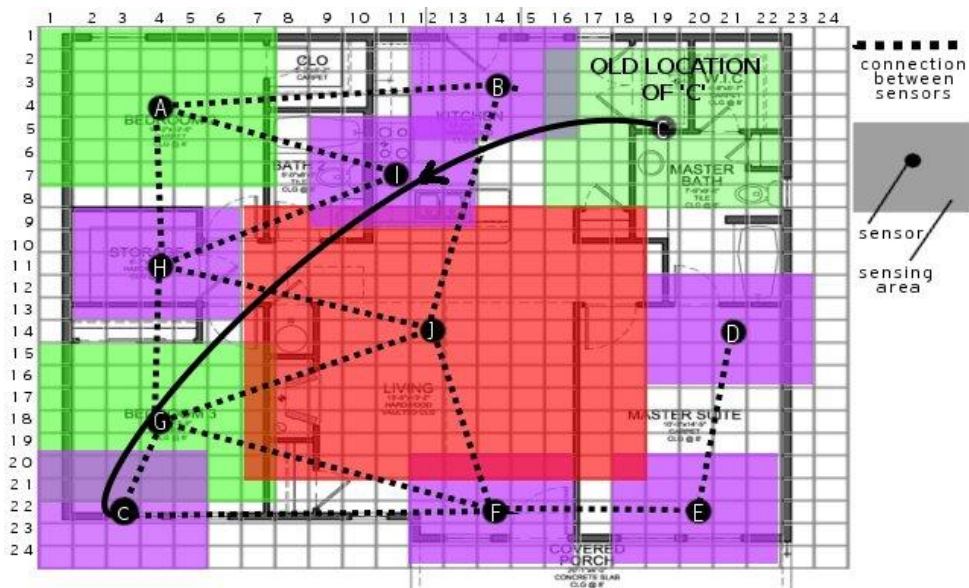


Figure 5-7: The Sensor "C" moving to new location.

All of the aforementioned nodes are used as a kind of proxy, to serve requests of users in their respective “cloaked areas”, allowing said users to utilize location based and other such enhanced services without disclosing their true identity and exact location. An enhanced view of the service’s overlay including users can be found in the figure below.

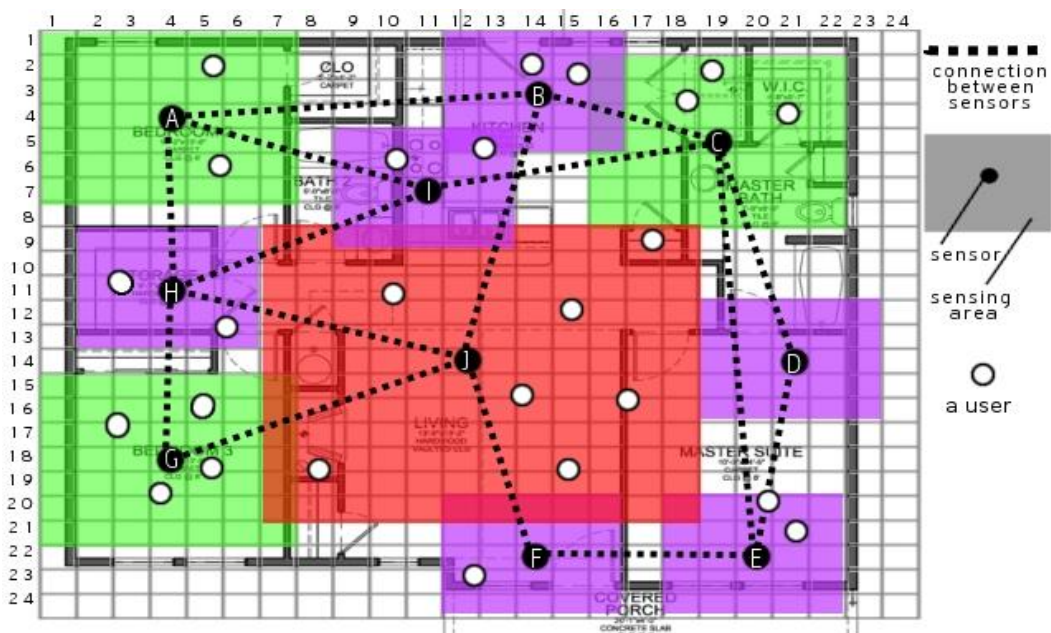


Figure 5-8: Service topology including users.

Every node communicates with neighbours sending them its name, sensing area and number of monitored users. The purpose of these lists is to allow nodes to find "K" users in an area as close as

possible. When peers receive these messages, they rebroadcast them until all their neighbouring sensor nodes have enough number of users, i.e., at least K users. A snapshot of peer lists pertaining to the example shown above could be the following:

A	B	C	D	E	F	G	H	I	J
H(2)	A(2)	B(2)	C(3)	D(0)	J(7)	J(7)	G(4)	C(5)	B(2)
I(2)	C(3)	D(0)	E(2)	F(1)	E(2)	H(2)	J(7)	A(2)	F(1)
B(2)		E(2)					I(2)		H(2)
							A(2)		

If a node does find the required number of users, it notifies its neighbours, otherwise it informs them that it is still trying to find K users. In turn, each neighbour sends its peer list to the node. When a node does reach the desired K level based on information received, it computes a score for every peer in its peer list. The score is defined as follows:

$$Score = \frac{Users\ in\ sensing\ area\ of\ node}{Distance\ of\ the\ node}$$

After the abovementioned score is calculated, the node selects the peers with the highest score and computes the cloaked area. The cloaked area is, therefore, a minimum bounding rectangle of the area (of the node and its chosen peers) which contains at least K users. Finally, the node sends the cloaked area it just calculated to its neighbouring nodes, validates that this area is unique and sends it to the server along with the total number of users contained in the area.

An example of this process can be seen below. In this example the anonymity level is assumed to be 8 (i.e. K=8).

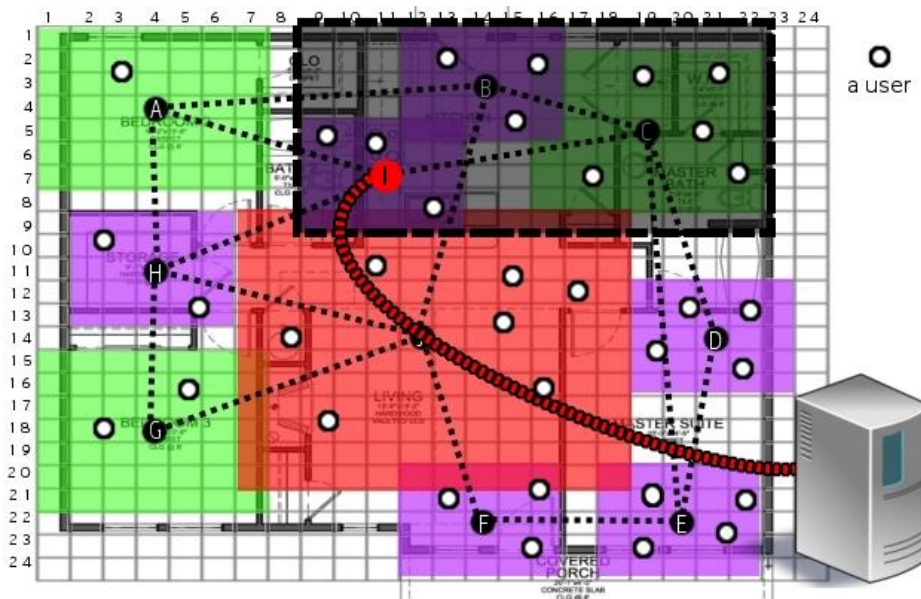


Figure 5-9: Example of cloaked area calculation.

Node "I" asks node "C" for more peers to reach the desired K-anonymity level. Node "C" responds by sending Node "B"'s info. This way node "I" can now reach the desire K level and computes a cloaked area, namely the rectangle defined by: (9, 1) - (22, 1) - (9, 9) - (22, 9). This area is validated as unique and transmitted to the server.

### 5.2.2 Implementation Details

Every node structure contains all the necessary information:

- Name
- Area-location
- User count
- Network address
- Listening port
- Send port

It also contains dynamic linked lists with information about its neighbours:

- A list of its neighbours
- Neighbour's cloaked areas
- A peer list of the neighbours that reached K anonymity level

The figures below detail all the data structures used. It should be mentioned that all data structures are dynamic to cater for the lack of memory on some of the nodes (i.e. nano nodes).

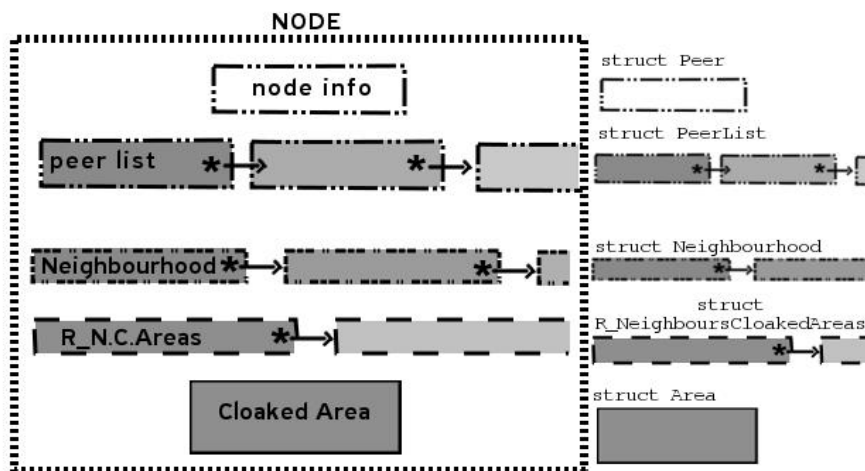


Figure 5-10: Node data structure.

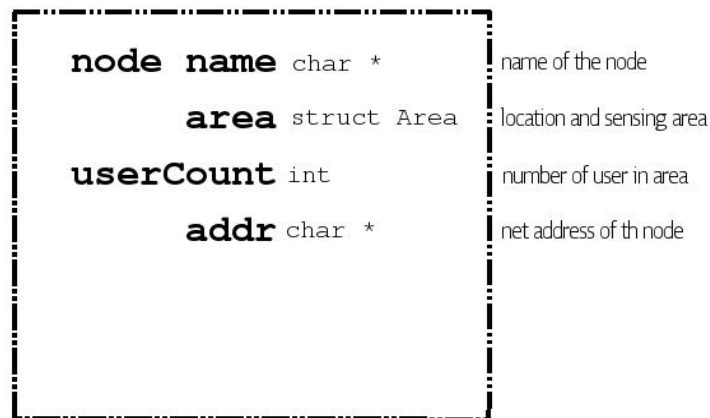


Figure 5-11: Peer structure.

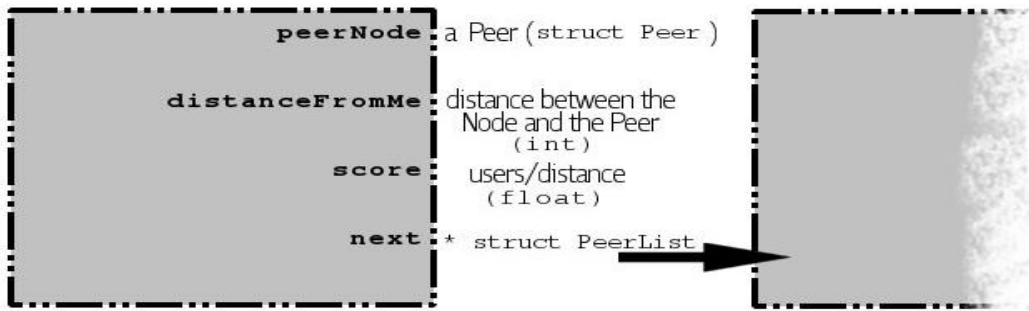


Figure 5-12: Peer list structure.

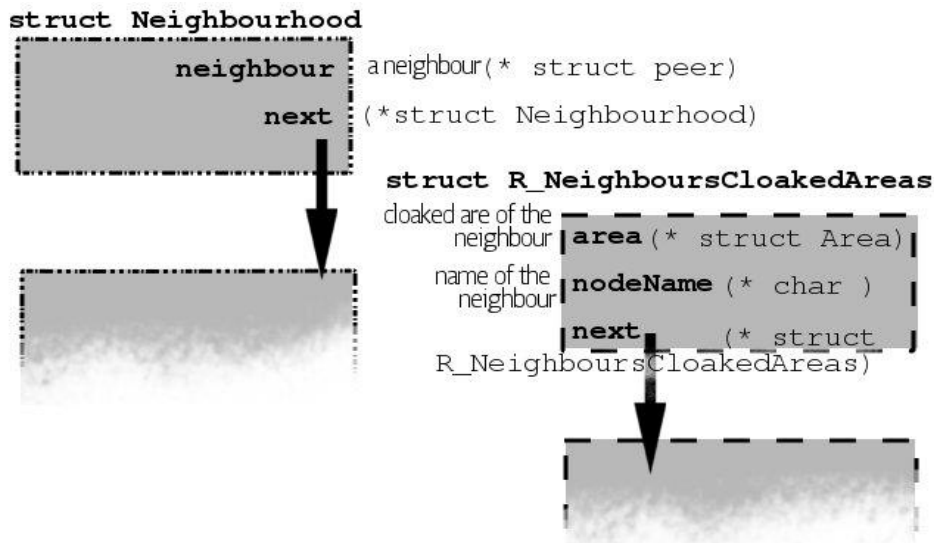


Figure 5-13: Neighbourhood and Neighbour’s cloaked area structures.

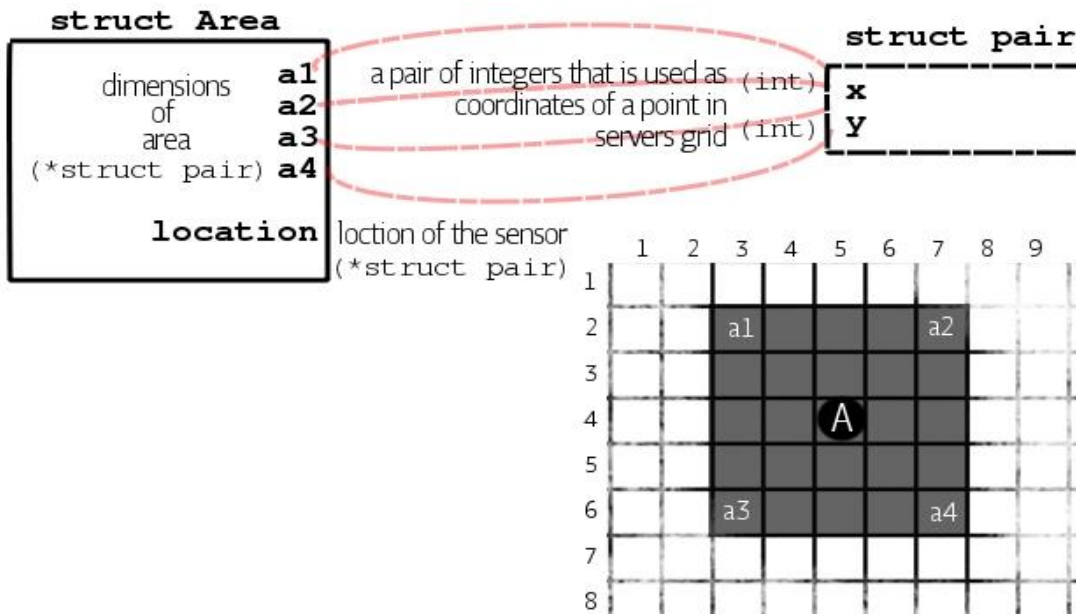


Figure 5-14: Area and Pair structures.

In the above figure, an example of how areas are calculated can also be seen. In the above case “A” node’s location would be point (5, 4) and his sensing area would be defined by points a1= (3, 2), a2= (7, 2), a3= (3, 6), a4= (7, 6).

### 5.2.3 Node Communication

A valid message interchanged between nodes must have one of the structured defined below:

1. /<callsign>\_<sender>/
2. /<callsign>\_<sender>\_<area>\_<userCount>/
3. /<callsign>\_<sender>\_<area>/

Callsigns types:

- “SNINF”: a node info package (type 2)
- “NOTIF”: Notification package. Sender reached K anonymity level (type 1)
- “CAREA”: Cloaked Area of the sender (type 3)
- “NOKUS”: Sender did not reach K anonymity level (type 1)

A typical node message exchange can be seen below.

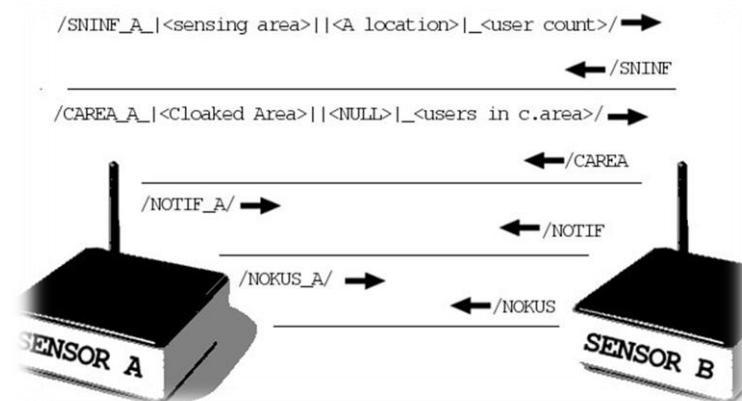


Figure 5-15: Typical node message exchange.

If the receiver accepts the message, it replies with the call sign. If the reply is valid, the sender considers that the message has been delivered successfully.

### 5.2.4 Server Platform

The server implementation was designed to be deployable in nSHIELD power nodes without imposing a significant performance overhead. The application was developed using Java and is deployed as a bundle for Knopflerfish, the service platform already present on nSHIELD power nodes. It features a graphical user interface (GUI) which displays the system grid, latest cloaked areas received and the pseudonym associated with those areas. A screenshot of the server interface can be seen in Figure 5-16.



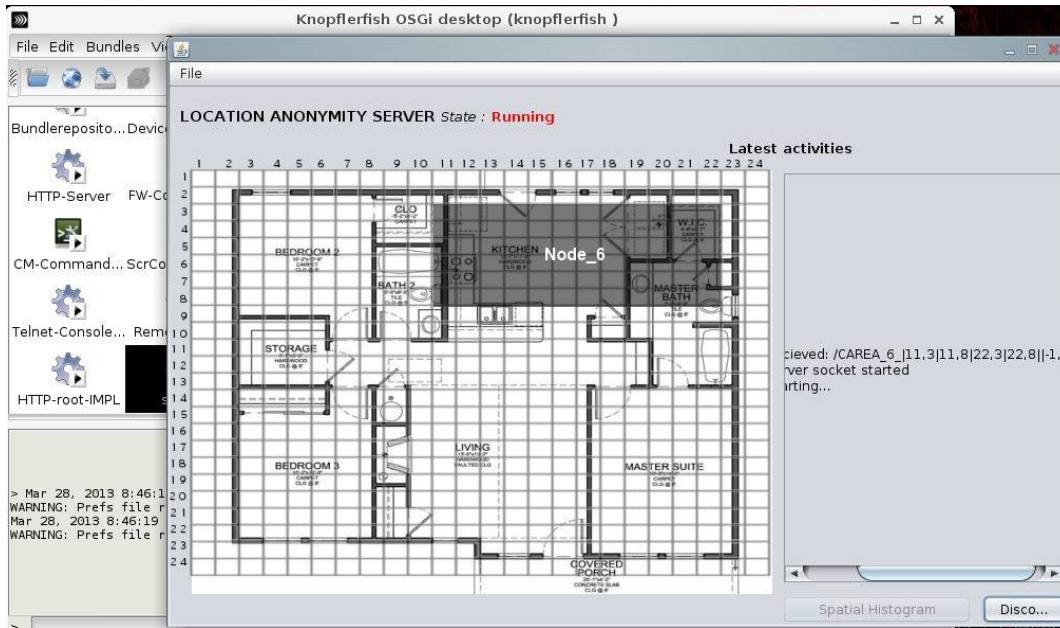


Figure 5-16: Server application. Cloaked area received from node “6”.

The main task of the server is to act as a proxy between the actual LBS service and the community of users and nodes. Hence, the server will be responsible for formulating service requests received by the nodes in a way that these will be compatible with the supported LBSs. E.g. if an LBS requires an explicit location to provide its services, the server can select a random position inside the cloaked area and report that position to the LBS server.

### 5.2.5 Anonymity Scheme Demonstration

The screenshots below detail a test execution of the resource-aware anonymity scheme implemented. In this demonstration two nodes were deployed, namely node “4” (at 192.168.1.4) and node “6” (at 192.168.1.6).

#### 5.2.5.1 Initialization

The nodes are initialized by the server, i.e. the server sends a list of neighbouring nodes to each node and its exact location in the server’s grid. Every node computes its sensing area and saves its neighbour list. Server also informs the nodes about the required anonymity level.

```
node info:
  name : 6
  area : |4,5|4,8|7,5|7,8||5,8|(!)
  usrCount : 5
  address : 192.168.1.6

Neighbours
-ODUMMY NODE
+1-home name : 4
  area : |2,3|2,6|5,3|5,6||3,4|
  usrCount : 2
  address : 192.168.1.4
NODE IS RUNNING...
```

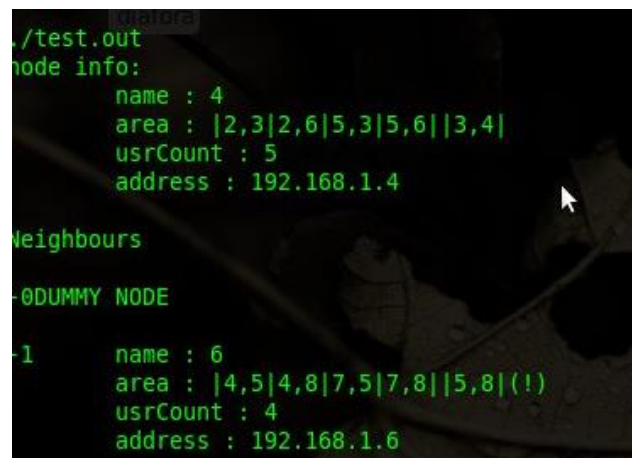
Figure 5-17: Typical Initialization of node “6”.

As can be seen above, the information displayed by the node after initialization includes:

- Name
- Sensing area and location : |<sArea>||<loc>|
- Users monitored at the time of initialization
- Network address

Moreover the peer list is displayed with all the relevant info known for each neighbour, since every node is initialized with a neighbours list. If a node changes position, the server sends to the node a new neighbours list that contains the new neighbouring nodes.

It should also be noted that the “(!)” mark displayed in the area field of the above figure is used to indicate that the area is not expected; it was chosen on purpose for this demo execution.



```
./test.out
node info:
  name : 4
  area : |2,3|2,6|5,3|5,6||3,4|
  usrCount : 5
  address : 192.168.1.4

Neighbours

0 DUMMY NODE

1  name : 6
   area : |4,5|4,8|7,5|7,8||5,8|(!)
   usrCount : 4
   address : 192.168.1.6
```

**Figure 5-18: Typical Initialization of node “4”.**

At this demo execution there are only two nodes, node “6” and node “4”.

### 5.2.5.2 Node Information exchange

The execution of the anonymity algorithm on node “6” is detailed in Figure 5-19, with an explanation next to the key output lines (see white text).

```

-----
:::START RESOURCE AWARE:::
-----
2013-2-27 13:53:15
- Anonymity level K      : 3
- Server address        : 192.168.1.6
- new users in my area  : 3
-2013-2-27 13:53:15 Broadcast 'Sensor Node INFo' to neighbours...
-
Send info to 4(192.168.1.4)...
cannot connect : Connection refused
ERROR !
-2013-2-27 13:53:16 Broadcast 'Sensor Node INFo' to neighbours...
-
Send info to 4(192.168.1.4)...
cannot connect : Connection refused
ERROR !
-2013-2-27 13:53:17 Broadcast 'Sensor Node INFo' to neighbours...
-
Send info to 4(192.168.1.4)...
cannot connect : Connection refused
ERROR !
-2013-2-27 13:53:17 Node info Recieved: sender : '4' area : '|2,3|2,6|5,3|5,6||3,4|' users : '1'
-2013-2-27 13:53:18 Broadcast 'Sensor Node INFo' to neighbours...
-
Send info to 4(192.168.1.4)...
Message accepted : '/SNINF_-' /SNINF_6_|4,5|4,8|7,5|7,8| |5,8| (!)_3/'
'Sensor Node INFo' SENT TO 192.168.1.4.
-2013-2-27 13:53:18 Notification recieved by 4
-----

```

-node '6' starts resource aware algorithm

-K ananymity level set by server

-At this demo execution server 'runs' at the same machine

-Node '6' monitored 3 users

-Node '6' broadcasts its info to neighbour nodes

-'6' tries to send info to node '4'

node '4' isn't connected yet (connection refused)

-while '6' was trying to send its info to '4', '4' sent its own info to '6'

-Node tries max 5 times to send its info , until reciever is ready . After .3 tries , '4' accepted this message :

Figure 5-19: Node "6" running the anonymity algorithm.

The same scheme is simultaneously run on node "4" (see Figure 5-20). The SNINF messages exchanged by nodes can also be seen in the screenshots.

```

-----
:::START RESOURCE AWARE:::
-----
2013-2-27 13:53:16
- Anonymity level K      : 3
- Server address        : 192.168.1.6
- new users in my area  : 1
-2013-2-27 13:53:16 Broadcast 'Sensor Node INFo' to neighbours...
-
Send info to 6(192.168.1.6)...
Message accepted : '/SNINF_-' /SNINF_4_|2,3|2,6|5,3|5,6||3,4|_1/'
'Sensor Node INFo' SENT TO 192.168.1.6.
-2013-2-27 13:53:17 Node info Recieved: sender : '6' area : '|4,5|4,8|7,5|7,8||5,8| (!)' users : '3'
-----

```

Figure 5-20: Node "4" running the algorithm.

### 5.2.5.3 Various K-anonymity states

As already mentioned, there are two different cases that have to be taken into account:

In some cases the node will manage to directly reach the desired K-anonymity levels, but in other instances some help from neighbouring nodes will be required.



### 5.2.5.3.1 Node reached required K-anonymity level

The successful state can be seen in the following figures, for nodes "6" and "4" respectively. Once a node finds K users, it broadcasts a notification to its neighbours.

```

-2013-2-27 13:53:15 ::k reached ,send notifications to neighbours
-2013-2-27 13:53:19 Broadcast'Reached K NOTIFICATION' to neighbours...
-
Send Notification to 4(192.168.1.4')...
Message accepted : '/NOTIF'-'/NOTIF_6/'.
'NOTIF BROADCAST'SENT TO 192.168.1.4.
ora
radamy)
-2013-2-27 13:53:15 -PEER LIST :
DUMMY NODE
  distance:0
DE 7.2.1
  name : 4
  area : |2,3|2,6|5,3|5,6||3,4|
  usrCount : 1
  address : 192.168.1.4
  distance:4
(WARNING : distance = 0 means test execution or error)
-2013-2-27 13:53:15
-SCORES :
--Node : '4' ,score : '0.25'.

```

-node '6' reached anonymity level K and broadcasts a notification to neighbour nodes

-Node prints its peer list for verification .

-Prints score list too . Score is userCount/distance. Every node computes a score for every peer and chooses the peers with the biggest score to compute a cloaked area.

Figure 5-21: Node "6" achieved the desire K-anonymity level.

```

-2013-2-27 13:53:17 Broadcast'Reached K NOTIFICATION' to neighbours...
-
Send Notification to 6(192.168.1.6')...
Message accepted : '/NOTIF_6'-'/NOTIF_4/'.
'NOTIF BROADCAST'SENT TO 192.168.1.6.
-2013-2-27 13:53:16 -PEER LIST :
DUMMY NODE
  distance:0
  name : 6
  area : |4,5|4,8|7,5|7,8||5,8|(!)
  usrCount : 3
  address : 192.168.1.6
  distance:4
(WARNING : distance = 0 means test execution or error)
-2013-2-27 13:53:16
-SCORES :
--Node : '6' ,score : '0.75'.
-2013-2-27 13:53:18 Notification recieved by 6

```

also '4' reached K anonymity level (3) . so it sends a notification to its neighbours .

-Notification was accepted by '6' . \*

-prints its peer list

-computes score for every peer

'4' recieved notification from node '6'.

Figure 5-22: "4" reached the required k-Anonymity level.

### 5.2.5.3.2 Node did not reach required K-anonymity level

The unsuccessful – in terms of k-Anonymity - state for node “6” can be seen below.

```

Message accepted : '/SNINF'-'/'SNINF_6_|4,5|4,8|7,5|7,8|5,6|3,4|_1/'.
'Sensor Node INFO'SENT TO 192.168.1.4.
-2013-2-27 15:52:31 '4'(192.168.1.4) didn't reach 'k' level anonymity (3)
Send him my peer list...
-2013-2-27 15:52:31 Send my peer list...
-2013-2-27 15:52:31 Peer list sent successfully
-2013-2-27 15:52:29 :::No k users on my sensing area
-2013-2-27 15:52:32 Broadcast 'NO K Users Notification'...
-Send No K Users Notification to 4(192.168.1.4)...
Message accepted : '/NOKUS'-'/'NOKUS_6/'.
'NO K Users'SENT TO 192.168.1.4.
-2013-2-27 15:52:32 Node info Recieved:'sender : '6' area : '|4,5|4,8|7,5|7,8|5,6|3,4|_1/' users : '1
  
```

Figure 5-23: Node "6" cannot find K users, thus node "4" sends its peer list to node "6".

If a node cannot find K users on its sensing area and its neighbours’ sensing area, it sends a “No K Users notification” (“NOKUS” message) and expects to receive their Peer list. The node that received a NOKUS message sends its peer list to the sender node in “SensorNodeINFo”-like messages for every peer on its peer list.

Once a node eventually reaches K-anonymity it’s ready to compute its cloaked area, validate it and inform its neighbours and server.

### 5.2.5.4 Error messages

A typical error situation can be seen in the figure below. In this example node “4” did not reach the required K level and therefore “6” sent its own peer list to “4”. Since “6” did not get a valid response from “4”, it assumes that node “4” did not receive the message.

```

-2013-2-27 13:59:1 Send my peer list...
Send info to 4(192.168.1.4)...
Warning : reply from server not as expected : '@F[+]-'/'SNINF_4_|2,3|2,6|5,3|5,6|3,4|_1/'.
ERROR !
-Send ERROR! Couldn't send Peer List to '4'
'6' sent /SNINF_4_.../ to '4' as '4' didnt reach K. '4' should reply
'/SNINF', so '6' determined that '4' didnt recieve the message.
  
```

Figure 5-24: Typical error message.

## 5.3 DDoS Attack Mitigation on SPD Power/Micro Nodes

### 5.3.1 Introduction

Distributed Denial of Service attacks are one of the most common weapons in cyber-crime. The evolution of computer viruses the last two decades shows a clear trend from destructive viruses that used to corrupt files and delete hard disks to stealthy parasites that once they infect a system, they use them to spread spam, host phishing web sites or make them part of a DDoS attack network also known as “botnet”. We

now face the reality of botnets that hold millions of compromised systems ready to cripple almost any network or internet service they choose.

During a DDoS attack, a massive amount of traffic arrives to the target of the attack (victim). The target is either a network service or the network itself. The victim's services are disrupted due to this large traffic or the computational overhead that it produces. This traffic is generated from thousands of compromised systems that belong to, probably, unaware users.

Due to the specification of the IP protocol, those systems can generate any kind of packets. Those packets can contain any random source IP address. Packets that originate from compromised systems as part of a DDoS attack usually hold a random source IP address or an address that belongs to another host. This technique, called IP spoofing, is one of the reasons that make DDoS attacks very difficult to be traced and/or countered. The victim accepts incoming traffic from irrelevant sources and any kind of filtering based on the source IP address is not possible without filtering out legitimate customers or users too. Either way, the final result is that the victim cannot provide its services on an acceptable rate or cannot provide them at all.

nSHIELD's power and, to an extent, micro nodes have the required characteristics to be considered as service providers. This brings those systems to the same environment with conventional service providers and makes them potential targets of DDoS attacks.

In order to be able to defend against DDoS attacks, we need an effective way of knowing the true source of an incoming packet or, at least, some indication that will enable us to incorporate an effective filtering strategy.

In this chapter we present two packet marking schemes that enable the victim to perform real time filtering of incoming DDoS traffic as well as trace this traffic to its true source. Given those schemes, we describe in detail the architecture of the filtering and traceback mechanism that is used on the victim's network.

The three basic methods that exist to date regarding IP traceback are ingress filtering [31], packet logging [32] and packet marking [33], [34], [35], [36], and [37].

Ingress filtering [31] dictates that each router should know the IP address space that each router's local interface is serving. When a packet arrives to the router's ingress interface it should have a valid IP address or it is dropped. This method is quite simple and effective for preventing IP spoofing but has its drawbacks. It requires considerable additional knowledge from the routers. It also depends on global deployment. Even today, a considerable percentage of the networks do not employ ingress filtering or similar methods and allow spoofed packets to travel through the Internet [38].

In packet logging, the routers keep logs regarding the packets that pass through them. With the help of those logs, a recent packet can be traced back to its original source. Routers are required to keep considerable amount of information especially in high bandwidth networks. In [32], we see that the memory overhead can be reduced by storing only a digest of the packet's header. Global deployment is also an issue in this method.

Packet marking was first introduced in [34]. In packet marking, the routers overload parts of the IP header of the traversing packets in order to put a marking that notifies the recipient of the packet of their presence on the route. The recipient gathers those markings and rebuilds the complete path that this packet traversed.

In probabilistic packet marking (PPM) [34], [35], the marking procedure is performed once every  $n$  packets. This reduces the computational overhead of the marking but increases the number of packets needed to reconstruct the path. In deterministic packet marking (DPM) [33], the marking procedure is performed for each packet at edge routers only. This reduces the number of packets needed for path reconstruction.

### 5.3.2 Packet Marking Schemes

The basic principle of our packet marking schemes [36], [37] is that, in order to be able to effectively stop an ongoing DDoS attack, we have to be able to distinguish which incoming packets are parts of the attack. We have to be able to perform this in real time so that we can filter each and every packet that belongs to the attack before it reaches the victim's server.

Existing mechanisms can identify the packets that belong to an ongoing DDoS attack, given enough packets and time to process them [39], [40], [41]. If the source IP address was reliable, they could filter all the packets that arrive from the IP address and stop the attack. Since this is not the case, packet markings are to be used instead of source IP address.

According to our schemes, the routers along a packet's path inject inside the IP header of that packet information that can be used as an indication regarding to the true source of this packet. Once the packets arrive at their destination, the victim can use this piece of information instead of the source IP address in order to distinguish which packets belong to which network. Moreover, our packet marking schemes enable real time filtering of the traffic that belongs to a DDoS attack mainly due to the fact that the filtering procedure is based, unlike most existing packet marking schemes, only on the information that exists inside each individual packet. In more detail, the packet marking procedure can be described as follows:

Once a packet enters the network, the first router along its path overwrites certain fields of the IP header and puts part of its IP address, that we will call "Router Signature", in that place. It also initializes a distance counter, that we will call "Distance Field", to zero. The distance field and the router signature occupy the part of the packet's IP header that we have overwritten. More detail about this encoding can be found in the next subsections.

Each subsequent router along the packet's path increases the distance field by one and puts the result of a <xor> between the existing router signature (inside the packet's header) and its own signature to the corresponding field inside the packet's header.

From the victim's point of view, all the incoming packets bear inside their IP header a distinctive marking that denotes their true origin. The DDoS detection system as well as the filtering mechanism and firewall can use those marks instead of the source IP address in order to decide, in real time since no other piece of information is required, which packets should be dropped. This scheme is able to provide information up to the closest router to the source. Thus when one or more hosts, from a certain network, participate in an ongoing DDoS attack, all the packets from that network will be dropped by the victim. This is an inherent limitation of any packet marking scheme. The more extensive the deployment of the marking scheme becomes, the more precise the filtering capabilities are.

In case there is a need of finding the true source of the attack packets, the victim needs to collect all those marks that participated in the attack and perform, recursively, the reverse marking procedure [37]. For the traceback procedure to work, it is required from the victim to hold an updated map of all upstream routers. That kind of map can be obtained by using standard trace route tools such as Skitter [42] that can map thousands of nodes every day in a non-intrusive manner. Furthermore, such a map can be obtained after the DDoS attack because of the fact that the traceback procedure, being recursive, is too computational intensive to be performed during an ongoing attack.

The marking procedure introduces no additional network bandwidth overhead, unlike packet logging schemes [32], since no control traffic is generated during the marking, filtering or traceback procedure. The computational overhead required by the routers is minimal and it involves one increment and one <xor> operation between constant values. The majority of the computational overhead is distributed amongst the victims of the DDoS attacks and not the network backbone itself. No additional memory overhead is introduced to the routers either.

The filtering procedure can be performed in real time since all the required information lies inside each individual packet's header unlike Probabilistic Packet Marking [34] that needs 500-4000 packets and Deterministic Packet Marking [33] that needs 7-10 packets to identify the source of one packet.



5.3.2.1 Scheme I

In the packet marking scheme I [36] we use the shaded part of the IP header that can be found in the figure below for the packet marking. From those 17 bits, we use 12 bits for the “Router Signature” field and 5 bits for the “Distance” field.

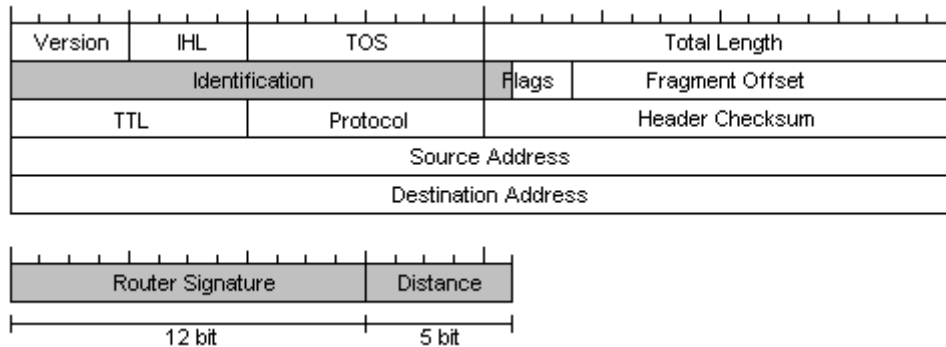


Figure 5-25: The IP header in packet marking I. In packet marking scheme I, we overload the shaded part of the IP header to put the 17 bit packet marking.

In every detection system exists the probability to classify a host that is not part of a DDoS attack as an attacking host (false positive) or an attacking host as legitimate (false negative).

The packet marking scheme is supposed to be deployed along with a DDoS detection mechanism. This mechanism identifies the sources of the DDoS attack as long as the information about the source of each packet provided by the marking scheme is correct. By design the packets that come from a host that participates on the attack will bear the same marking. In the rare exception that the routing changes during the attack, either this host will disappear, from the victim’s point of view, and be replaced by another host (the marking of the host will change) or other hosts will appear as parts of the DDoS attack (some of the host’s traffic will follow another route). Nevertheless the marking scheme produces no false negatives and the false negative probability of the combination of the two systems depends only on the detection system.

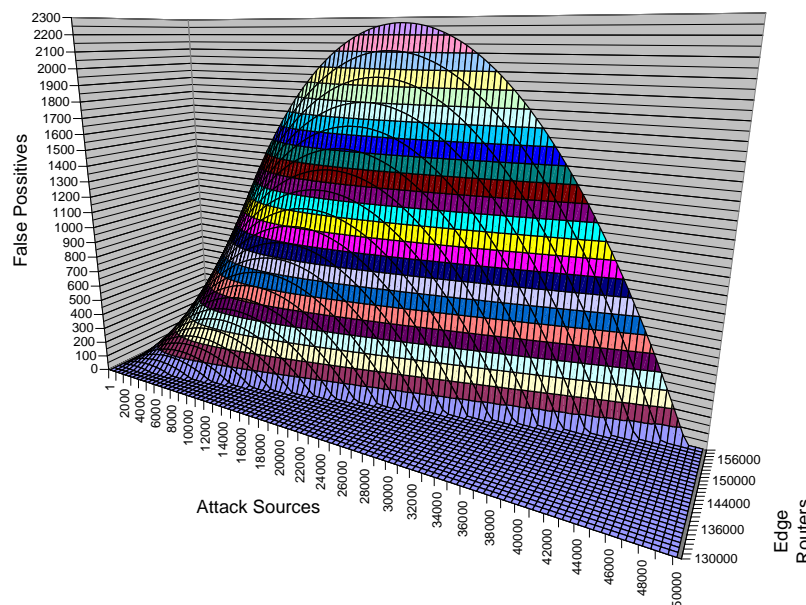


Figure 5-26: The false positives of packet marking scheme I for 130 to 150 thousand edge routers. As the attack sources rise, we notice a considerable decline in false positives.

The false positive probability depends on the number of the attackers, the total number of edge routers and the length of the marking field. Let R be the number of edge routers and A the number of attacking hosts. Let n be the length of the marking field. The number of edge router IP addresses that match a specific marking is  $\mu=R/2^n$ . The number of distinct markings that the attacking packets will bear is the number of the resulting faces of a  $2^n$  faced die after A throws. The latter is a special case of the occupancy problem [43]. Thus the expected number of distinct markings of the attack packets is:

$$M = 2^n - 2^n(1 - 1/2^n)^A$$

The number of false positives F can be calculated as follows:

$$F = M * \mu - A$$

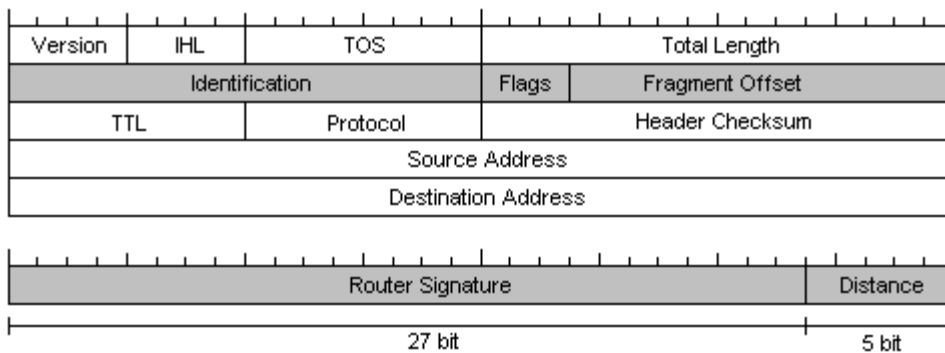
As the number of attackers increase, the collisions between sources that belong to the attack also increase. Thus the percentage of false positives lowers for large numbers of attacking hosts as shown in the figure above.

**5.3.2.2 Scheme II**

In packet marking scheme II [37] we use the shaded part of the IP header that can be found in the figure below for the packet marking. From those 32 bits, we use 27 bits for the “Router Signature” field and 5 bits for the “Distance” field.

In order to ensure that the packet will not be corrupt due to the IP fragment fields overload, before the packet reaches its destination, the offset and flags fields are set to zero. This is done by the closest to the destination router which is usually the one that collects the markings too. All the packets except fragmented packets will remain compatible with the IP specification. The fragmented packets will still be corrupt though.

The fact that we use 27 bits for the router signature field ensures that there is enough space to cover all the class A and B networks as well as most of the class C networks. The routers simply put their 27 most significant bits of their IP address into the field during the marking procedure. Therefore the marking procedure has virtually zero fault positive probability unlike previous marking schemes [33], [34], [35], [36]. False negative probability is the same as in packet marking I.



**Figure 5-27: The IP header in packet marking II. In packet marking scheme II, we overload the shaded part of the IP header to put the 32 bit packet marking.**

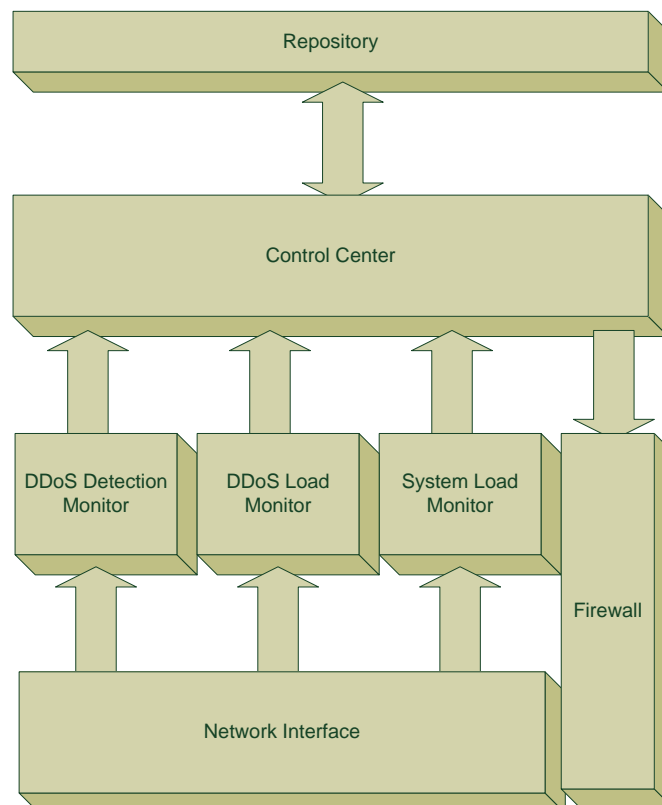
**5.3.3 Filtering and Traceback Mechanism**

Given the deployment of one of the above packet marking schemes; the victim is in the position of knowing valuable information about the origin of each incoming packet. When a network or system is under a DDoS attack, two things are of highest priority. First, the victim should be able to keep its service online and serve its legitimate customers with as little disruptions as possible. Second priority is to be able to identify the true sources of this attack and see what it can be done to stop it permanently.

In order to achieve undisrupted services, the victim must be able to filter out in an intelligent manner most of the incoming attack traffic. The fine line lies to the point where the victim drops enough attack traffic to hold the server load to acceptable levels so that it can serve legitimate requests. In most cases, dropping attack traffic, which means putting entire networks to a black list and refusing any kind of traffic originating from them, results in some legitimate clients to be denied access too. This is the result of two factors. The first factor is the inherited false positives of the packet marking schemes which indicate that some networks that are not part of the attack could be marked as such. The second and most important, due to the existence of packet marking scheme II, factor is the inability to distinguish nodes inside the same network. This means that there are situations where legitimate clients and attack nodes lie inside the same network. Refusing incoming traffic from that network, even if we know for sure that is one of the origins of the DDoS attack, means that some legitimate users will not be able to access the service.

Filtering the incoming traffic should be done in real time for each packet. The factor that enables real time filtering is the marking that lies inside each packet header. Once it is known that packets from a certain network (bearing a distinct marking) are part of the DDoS attack, all those packets can be easily identified from their marking and be dropped. The lack of any kind of necessary correlation between packets and the complete independency from the source IP address ensure that the filtering mechanism can perform in real time and be robust against IP spoofing respectively.

Apart from filtering, it is often necessary to identify the exact address of the networks that took part in the DDoS attack so that the victim can either inform the administration of those networks for the existence of a live botnet inside their networks or demand compensation. Packet markings do not show the exact address of the incoming packet but this can be calculated following the traceback procedure that has been described in previous sections. This procedure is time consuming but can be followed during or after the DDoS attack in a separate environment. All the necessary information is stored inside the mechanism's repository and, due to its tiny storage size, it can be held indefinitely.



**Figure 5-28: The filtering and traceback mechanism architecture.**

In the figure above we show the architecture of the filtering and traceback mechanism. The mechanism consists in a DDoS load monitor, a DDoS detection monitor which can be part of existing network intrusion detection (NIDS) system, a system load monitor, and a reconfigurable firewall able to handle packet markings, a repository and a control centre.

In general, the two DDoS monitors observe incoming traffic and send alerts to the control centre. The control centre gets periodic reports from all three monitors. The repository holds information about the networks that are of interest to the victim's company or organization. In the case of an ongoing DDoS attack, the control centre evaluates the reports and, in conjunction with the information in the repository, sends directives to the firewall regarding which networks (i.e. markings) should it filter out.

In the following sections, we will describe each module and we will present some test case scenarios.

### 5.3.3.1 Module Description

The modules of the filtering and traceback mechanism can be seen in the figure above. The system, implements a reconfigurable firewall that handles all the filtering according to the directives taken from the control centre.

As the packets travel from the network interface, they are being observed by two DDoS monitors. The first is the DDoS detection monitor who determines whether or not the system is under DDoS attack and also informs the control centre which networks are parts of this attack. Once the attack alert has been sent, the markings of the identified networks are being sent to the control centre the moment they are observed. The DDoS detection monitor can be part of a more complete NIDS system that is probably already installed on the victim's system. It uses any of the existing DDoS detection mechanisms [39], [40], [41] which incorporate detection methods that span from statistical anomaly detection to neural networks and pattern matching mechanisms or the DDoS detection system designed in nSHIELD. One major difference from conventional monitors is that all the necessary packet analysis and correlation is being done based on the existing packet marking instead of the source IP address. Detection mechanisms usually involve large samples of packets originating from the same source and/or deep packet inspection. This procedure is not required to be in real time as long as it produces results with low positive or negative fault probability. The actual filtering is being performed by the firewall once the monitor sends the results to the control centre.

The second DDoS monitor gathers load statistics regarding the attack traffic. Once a DDoS attack begins, it receives the offending markings from the control centre and it sends back the traffic load that each attack source (network) produces. This information helps the control centre to decide which network should be filtered out, depending on the overall server load.

The last monitor is a standard system load monitor that reports back to the control centre. This monitor also triggers a DDoS attack alert when the server load reaches a certain use specified threshold. This alert is indicative of the possible existence of an ongoing attack but the absence of a similar alert from the DDoS detection monitor in a user specified amount of time results in a visual alert and/or report to the network administrator.

The repository is a database the holds information about the networks that are of interest to the server's company or organization. Each record has the fields that can be seen in the table below.

The fields "network name", "network", "network marking", "average traffic" and "block" are standard while the rest can be user specific. Each user specific field reflects one metric that should be considered when deciding to block this network as part of an ongoing attack. Such fields also have a weight factor. This factor denotes the percentage that this characteristic should take part to the final result.

The necessity of a repository occurs due to the fact that, in many cases, networks that are part of an ongoing DDoS attack host legitimate users too. It is therefore necessary to hold back on blocking certain networks until it is absolutely necessary. The combined results of the user specific factors along with their weights for each network give to the control centre the priority list that dictates the order that those networks get blocked.



The “average traffic” field is used to calculate the approximate “real” attack traffic that originates from this network if this network is part of an ongoing DDoS attack. The “block” field denotes a network that is either of no value at all to the organization or, for some other reason, should be blocked even before attempting to block unknown networks which usually have the first priority.

**Table 5-2: Each record in the repository denotes a network and has a set of standard fields (those marked with \*) and user specific fields along with their respective weights.**

Id*
Network Name*
Network*
Network Marking*
Network Size
Network Size Weight
Customer Size
Customer Size Weight
Customer Priority
Customer Priority Weight
Region
Region Weight
Average Traffic*
Block*

The next module is the firewall. Apart from the usual features, the firewall can be reconfigured on the fly by the control centre as more markings appear in the firewall’s black list. This type of filtering is based only on the marking that the packet bears and not its source IP address. The remaining firewall rules can use the usual method of source identification. The firewall also strips the marking from the packet and resets the overloaded IP header fields to their default values depending on which packet marking scheme is used.

The last component is the control centre. The control centre is the component that gathers all the data from the three monitors and decides which networks should be blocked. This decision is based on the monitor readings and the information that lies inside the repository regarding known networks.

In the case where the DDoS attack originates from unknown networks (not present in the repository), it progressively blocks those networks by sending their respective markings to the firewall’s black list. The readings from the DDoS load monitor enable the control centre to randomly block the minimum amount of attacking networks that will bring the server load to comfortable levels in the first iteration. In subsequent iterations, possible fluctuations on the traffic rate will result in more or less networks blocked respectively. The selective blocking of even unknown networks ensures that possible legitimate users from those networks get to be served if the server load permits it.

When the DDoS attack involves networks that exist in the repository, it follows the following procedure: first it blocks the networks that have the “block” field enabled. Then it uses the readings from the DDoS load monitor to determine the server load in the case where every network (known or unknown) is blocked. If the result is lower than the user defined “comfortable” level, it starts subtracting known networks from the candidate black list according to their position in the repository’s priority list until the projected load reaches as close to the user defined maximum load. In the case where more than one combination exists, i.e. more than one known networks of the same priority can fit in the last place before the projected server load reaches the maximum, the “average traffic” field is used instead of priority. If the result is still lower than the “comfortable” level, unknown networks are being subtracted from the candidate black list. The resulting list is sent to the firewall which handles the filtering procedure. In each subsequent iteration, measurements from the server load and DDoS load monitors, are being taken to

determine if further reconfiguration is needed. If the server load exceeds the upper or lower threshold or the traffic load of the attacking networks changes considerably, the control centre repeats the procedure.

This procedure ensures service availability while taking care of blocking the least possible legitimate users.

### 5.3.3.2 Scenarios and Test Cases

A typical scenario of a DDoS attack starts with an alert from DDoS detection monitor and a significant rise of the system load as perceived from the respective monitor. Let's say that the attack originates from five distinct networks. After a short amount of time, the control centre receives the markings of those five networks from the DDoS detection monitor. It performs a repository lookup and finds that two of those networks are large ISPs with considerable customer base that should be handled with different priorities. The control centre gets readings from the DDoS load monitor and based on those readings and the known networks priorities tries to fit as many networks as possible inside the 80-100% server load boundary that has been configured by the administrator. The result is that the three unknown networks get blocked at first. Of the remaining two networks, the one with the lowest priority gets blocked too. Having a projected system load of ~70% and not being able to fit the other known network, the control centre chooses not to block the one of the three unknown networks that produces traffic equal to 15% of the server capacity. The server load stabilizes to ~85% for some time until the DDoS attack stops. At this moment, the control centre revokes all filtering directives. Two more test cases can be seen in the figures below.

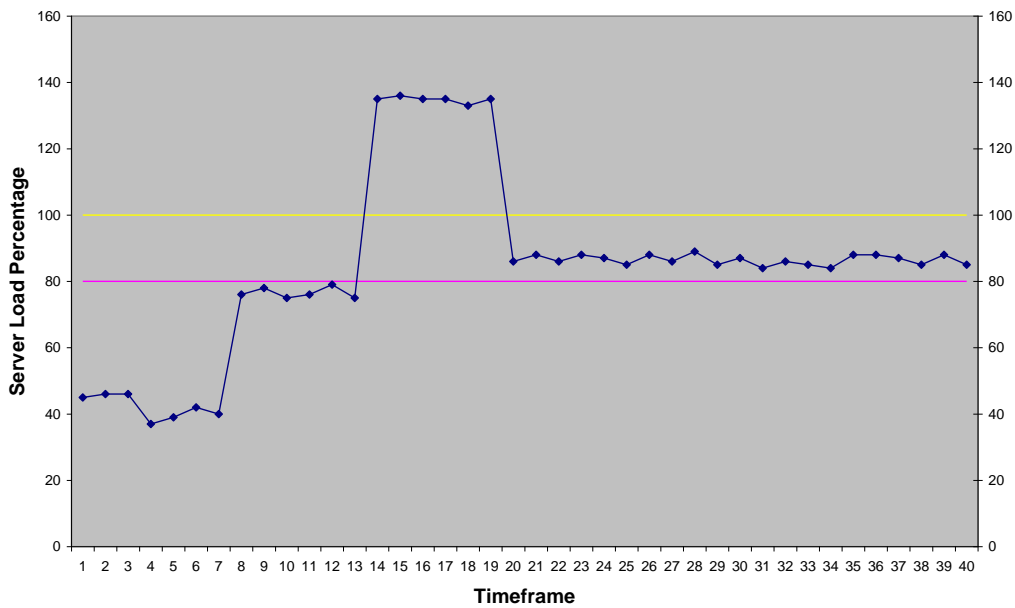
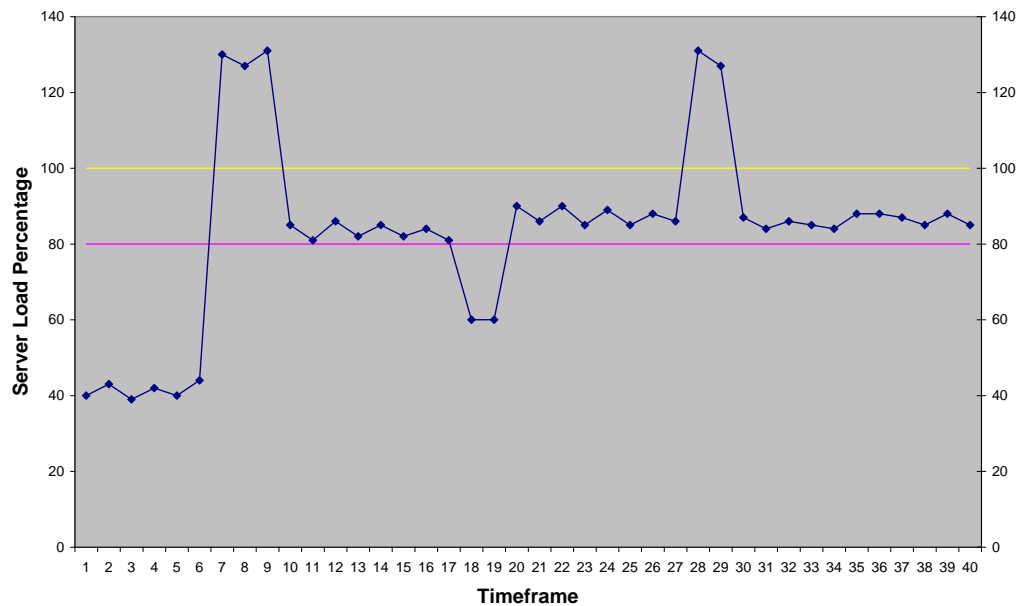


Figure 5-29: Test case 1. A DDoS attack starts progressively. The control centre responds and stabilizes the server load to a comfortable level.



**Figure 5-30: Test case 2. This DDoS attack incorporates networks that are of value to the victim's organization. Some networks stop sending traffic in the middle of the attack while others join the attack later.**

In the first figure we see at timeframe 7 the start of a DDoS attack. At timeframe 14, the server load exceeds server capacity. Until then, no action was taken since the server load was under the upper threshold. After a while, at timeframe 20, the control centre has received the markings of all attacking networks and chooses to block some of them randomly since none of them had a record in the repository.

In the second figure we see the start of a massive DDoS attack at timeframe 6. This time the attack originates from networks that are known to the victim's repository as well as unknown networks. The control centre receives all the markings and each network's approximate traffic and takes the decision to block all unknown networks and one known that has low priority, so that the server load comes to acceptable levels. At timeframe 17, some of the attacking networks stop sending traffic. The control centre then releases the one known network that was previously blocked as well as a few unknown networks. At timeframe 27, more networks join the ongoing DDoS attack. The control centre repeats the decision procedure and stabilizes the system load.

### 5.3.4 Conclusions

We designed a packet marking scheme in conjunction with an intelligent filtering and traceback mechanism that can effectively stop ongoing DDoS attacks. We described in detail the architecture of a highly configurable mechanism able to react in different attack scenarios and ensure the highest amount of legitimate user service under an ongoing DDoS attack. With precise tuning of this mechanism, an organization can provide robust while flexible protection against such attacks.

## 6 Cryptographic technologies

The main research areas targeted by cryptographic technologies are the following:

- Hardware and software crypto technologies (including asymmetric cryptography, ECC cryptography and symmetric cryptography suitable for constrained devices).
- Anti-tamper technologies.
- Key exchange mechanisms.

### 6.1 Elliptic Curve Point Multiplication over Prime Fields Library

This library represents the implementation of the point multiplication operation on an elliptic curve on prime field, which is the core of elliptic curve-based cryptographic protocols like Elliptic Curve Diffie-Hellman (ECDH) and Elliptic Curve Digital Signature Algorithm (ECDSA).

Detailed information about the algorithm and the theory included in the library can be found in the deliverable D3.1.

#### 6.1.1 Description

The elliptic curve point multiplication operation is defined as  $Q=kP$ , where  $Q, P$  are points in a previously chosen elliptic curve defined over a prime field  $GF(p)$ , and  $k$  is a field element. Here  $p$  is the characteristic of the field.

The elliptic curves supported are in the simplified Weierstrass form  $y^2=x^3+ax+b$ , where  $a, b$  are elements of  $GF(p)$  with  $p$  different from 2 or 3. Supported curves include those specified by NIST (Standards for Efficient Cryptography 2 – September 20, 2000, available at [www.secg.org](http://www.secg.org)) indicated as secp160r2, secp192k1, secp192r1, secp224k1, secp224r1, secp256k1, secp256r1, secp384r1, secp521r1.

This library supports elliptic curve point representations called Affine coordinates and Jacobian coordinates.

This library employs the Montgomery Ladder algorithm in order to accomplish the elliptic curve point multiplication. This is the most efficient algorithm producing side-channel attacks-resistant cryptosystems. For testing purposes only, the elliptic curve point multiplication is provided with the trivial binary method also.

#### 6.1.2 Basic Interface

Configuration parameters: field characteristic  $p$ , elliptic curve coefficients  $a, b$  belonging to  $GF(p)$ .

Input parameters: scalar  $k$  belonging to  $GF(p)$ ,  $x$  coordinate of the affine point  $P(x, y)$  belonging to the elliptic curve  $y^2=x^3+ax+b$  defined over  $GF(p)$ .

Library output: if  $x$  is a valid coordinate of a point over the specified curve, the library returns the affine point  $Q = kP$ .

#### 6.1.3 Code Compilation and Dependencies

This library requires the GNU Multiple Precision Library (libgmp) version 4.3 or higher. This dependency is freely available at [44], with LGPL license.

The entry point of the library is represented by the source file `Lib.h`. A custom program including this source file can be compiled with the following line:

```
$ gcc -o CustomProgram CustomProgram.c Lib.c -lgmp -lm
```

### 6.1.4 Hardware platform

Hardware tests for present library are conducted on the OMBRA platform, provided by Selex ES, with a Linux operating system. This platform presents a SOIC integrating an ARM Cortex-A8 running at 1GHz.

### 6.1.5 Library API

- **void Montgomery(mpz\_t x3,mpz\_t y3,mpz\_t x1,mpz\_t y1,mpz\_t field,mpz\_t a,mpz\_t k):** performs  $(x_3,y_3)=k(x_1,y_1)$ , using the Montgomery algorithm, on a curve with parameter  $a$  and over the prime field. Point operations are performed in affine coordinates.
- **void Montgomery\_j(mpz\_t x3,mpz\_t y3,mpz\_t x1,mpz\_t y1,mpz\_t field,mpz\_t a,mpz\_t k):** performs  $(x_3,y_3)=k(x_1,y_1)$ , using the Montgomery algorithm, on a curve with parameter  $a$  and over the prime field. Point operations are performed in Jacobian coordinates.
- **int testCurve(mpz\_t delta,mpz\_t a,mpz\_t b,mpz\_t field):** checks if the elliptic curve described by the parameters  $a, b$  over the prime field is supersingular, returning  $\text{delta}=4a^3-27b^2$ .
- **int testPoint(mpz\_t y\_square,mpz\_t a,mpz\_t b,mpz\_t field,mpz\_t x):** checks if field element  $x$  is a valid coordinate on the curve described by  $a, b$  returning corresponding  $y^2$ .
- **int Tonelli(mpz\_t c,mpz\_t a,mpz\_t p):** given a field element  $a$ , computes  $c^2=a \pmod p$ . Used to evaluate  $y$  from  $y^2$  provided by test Point API.
- **int Hasse(mpz\_t field):** returns the theoretical upper bound of the number of elliptic curve points over specified field.
- **void KappaP(mpz\_t x3,mpz\_t y3,mpz\_t x1,mpz\_t y1,mpz\_t field,mpz\_t a,mpz\_t k):** performs  $(x_3,y_3)=k(x_1,y_1)$  on a curve with parameter  $a$  and over the prime field. Point operations are performed in affine coordinates. This algorithm is only for testing purposes, since it makes the cryptosystem vulnerable to side-channel attacks.
- **void KappaP\_j(mpz\_t x3,mpz\_t y3,mpz\_t x1,mpz\_t y1,mpz\_t field,mpz\_t a,mpz\_t k):** performs  $(x_3,y_3)=k(x_1,y_1)$  on a curve with parameter  $a$  and over the prime field. Point operations are performed in Jacobian coordinates. This algorithm is only for testing purposes, since it makes the cryptosystem vulnerable to side-channel attacks.

### 6.1.6 Service functions

- **void sum(mpz\_t x3,mpz\_t y3,mpz\_t x1,mpz\_t y1,mpz\_t x2,mpz\_t y2,mpz\_t field):** sum of two elliptic curve points in affine coordinates over a specified prime field:  $(x_3,y_3)=(x_1,y_1)+(x_2,y_2)$
- **void duple(mpz\_t x3,mpz\_t y3,mpz\_t x1,mpz\_t y1,mpz\_t field,mpz\_t a):** doubling of an elliptic curve point in affine coordinates over a specified prime field and on curve specified parameter  $a$ :  $(x_3,y_3)=2(x_1,y_1)$
- **void sum\_j(mpz\_t X3,mpz\_t Y3, mpz\_t Z3,mpz\_t X1,mpz\_t Y1,mpz\_t Z1,mpz\_t X2,mpz\_t Y2,mpz\_t Z2,mpz\_t field):** sum of two elliptic curve points in Jacobian coordinates over a specified prime field:  $(X_3,Y_3,Z_3)=(X_1,Y_1,Z_1)+(X_2,Y_2,Z_2)$
- **void duple\_j(mpz\_t X3,mpz\_t Y3,mpz\_t Z3,mpz\_t X1,mpz\_t Y1,mpz\_t Z1,mpz\_t field,mpz\_t a):** doubling of an elliptic curve point in Jacobian coordinates over a specified prime field and on curve specified parameter  $a$ :  $(X_3,Y_3,Z_3)=2(X_1,Y_1,Z_1)$
- **void a2j(mpz\_t x,mpz\_t y,mpz\_t X,mpz\_t Y,mpz\_t Z):** conversion from affine point to Jacobian point:  $(x, y) \Rightarrow (X,Y,Z)$
- **void j2a(mpz\_t x1,mpz\_t y1,mpz\_t X1,mpz\_t Y1,mpz\_t Z1, mpz\_t field):** conversion from affine point to Jacobian point:  $(X,Y,Z) \Rightarrow (x, y)$

### 6.1.7 Code Samples

An example of library usage is provided in Main.c: this program performs a simple Elliptic Curve Diffie-Hellman protocol. The example code shows how to initialize the memory for every finite field element (mpz\_t type from libgmp), the definition of some domain parameters of the protocol (underlying prime field characteristic, elliptic curve parameters, generator point affine coordinates) and the actual protocol execution.

## 6.2 Compact Crypto Library

Well-known crypto libraries, like open SSL, target on mainstream applications. Other libraries that are designed for embedded system applications contain redundant functionality as they support a wide range of cryptographic primitives. For nSHIELD cryptographic technologies, we implement a compact crypto library for a subset of lightweight ciphers and compact implementations of standard cipher. The library utilizes open source implementation of known ciphers. We implement a common API for utilizing all of them with their different parameters.

### 6.2.1 Description

In this first prototype, the library contains block/stream ciphers and hash functions. For block ciphers, it supports AES [45], DES [46], 3DES [46], PRESENT [47], LED [48], KATAN [49], KTANTAN [49], Clefia [50], Camellia [51], XTEA [52] and XXTEA [53]. The block ciphers can operate in ECB, CBC and CTR modes of operation. The padding schemes that have implemented are the zeroPadding, PKCS5, PKCS7, ISO\_10126-2, ISO\_7816-4 [54] and X9.23. For stream ciphers, it supports the ARC4 [55] and the eSTREAM project [56] finalists Salsa [57], Rabbit [58], HC128 [59], SOSEMANUK [60], Grain [61], Grain-128 [61], Trivium [62] and Mickey v2 [63]. For hash functions, it supports the new SHA-3 function Keccak [64] and the other finalists of the SHA-3 contest [65] Blake [66], JH [67], Groestl [68] and Skein [69]. The crypto library is implemented in C language. We apply and test the library on on MemSic IRIS [70], BeagleBone [71], BeagleBoard [72] and BeagleBoard-xM [73] devices.

### 6.2.2 Basic Interface

Configuration parameters: the type of the crypto-primitive [block cipher, stream cipher, hash function]

Input parameters: The input parameters of each crypto-primitive. For block ciphers, the input parameters are the cipher name (AES, DES, TripleDES, PRESENT, LED, KATAN, KTANTAN, CLEFIA, CAMELLIA, XTEA and XXTEA), the encryption key, the plaintext/ciphertext, the mode of operation (ECB, CBC and CTR) and the padding (ZEROPADDING, PKCS5, PKCS7, ISO\_10126-2, ISO\_7816-4 and X9.23). For the stream ciphers, the input parameters are the cipher name (ARC4, HC128, RABBIT, SALSA20, SOSEMANUK, GRAIN, GRAIN128, TRIVIUM and MICKEY2), the encryption key, the initialization vector (IV) and the plaintext/ciphertext. For the hash functions, the input parameters are the hash function name (MD5, SHA-1, SHA-256, SHA-512, SHA-3, KECCAK, BLAKE, JH, GROESTL and SKEIN), the encryption key and the plaintext.

Library output: if the input parameters are valid, the library returns the result of the crypto-primitive. For a block/stream cipher encryption/decryption returns the ciphertext/plaintext. For a hash function returns the message digest. If the input parameters are invalid, the library returns error codes.

### 6.2.3 Code compilation and Dependencies

This library embodies open source implementations of crypto-primitives.

- The full library is compiled by the command: `$ make`

One can compile specific components of the library like:

- Compile all the block ciphers: `$ make block_ciphers`

- Compile all the stream ciphers: `$ make stream_ciphers`
- Compile all the hash functions: `$ make hash_functions`

Or even specific crypto-primitives:

- E.g. Compile the block ciphers AES and the stream cipher Salsa20:  
`$ make --enable-aes --enable-salsa20`

The entry point of the library is represented by the source file `crypto_lib.h`. A custom program must include the library header in its source file:

```
"include crypto_lib.h"
```

## 6.2.4 Hardware platform

Hardware tests were conducted on Memsic IRIS devices [70], with Contiki operating system. IRIS uses the Atmel ATmega 1281 processor at 8 MHz, with 8-KB RAM and 128-KB program flash memory. For nSHIELD's scope, Memsic IRIS is considered as a nano node. We also test the proposed library on the BeagleBone [71], BeagleBoard [72] and BeagleBoard-xM [73] platform, with an Ubuntu Linux operating system. For system in chip, the BeagleBone uses an AM3359, the BeagleBoard uses OMAP3530 and the BeagleBoard-xM uses a DM3730. All of them include an ARM Cortex-A8 single core CPU running at 500-720 MHz, 720MHz and 1GHz respectively. BeagleBone and BeagleBoard have 256-MB RAM and BeagleBoard-xM 512-MB RAM. For nSHIELD's scope, BeagleBone and BeagleBoard are micro/personal nodes and BeagleBoard-xM is a power node.

## 6.2.5 Library API

For block cipher en/decryption:

- **Byte \* cryptolib\_encrypt(char \*cipher\_name, byte \*plaintext, byte \*key, char \*mode, char \*padding):** encrypts a plaintext with the block cipher 'cipher\_name' and encryption key 'key', with operation mode 'mode' and padding 'padding'. The function returns the ciphertext.
- **Byte \* cryptolib\_decrypt(char \*cipher\_name, byte \*plaintext, byte \*key, char \*mode, char \*padding):** decrypts a ciphertext with the block cipher 'cipher\_name' and decryption key 'key', with operation mode 'mode' and padding 'padding'. The function returns the plaintext.

For stream cipher en/decryption:

- **Byte \* cryptolib\_encrypt(char \*cipher\_name, byte \*plaintext, byte \*key, byte \*iv):** encrypts a plaintext with the stream cipher 'cipher\_name', encryption key 'key' and initialization vector 'iv'. The function returns the ciphertext.
- **Byte \* cryptolib\_decrypt(char \*cipher\_name, byte \*plaintext, byte \*key, byte \*iv):** decrypts a ciphertext with the stream cipher 'cipher\_name', decryption key 'key' and initialization vector 'iv'. The function returns the plaintext.

For hash functions:

- **Byte \* cryptolib\_hmac(char \*hmac\_name, byte \*plaintext, byte \*key):** calculates the digest of a plaintext with the hash function 'hmac\_name' and the encryption key 'key'. The function returns the digest.

## 6.2.6 Service Functions

For input/output:

- **Byte \* cryptolib\_read\_file\_text(char \*plainFILE, int \*plbytes):** reads the file 'plainFILE' that contains ASCII characters. The function returns the data that was read in bytes.

- **Byte \* cryptolib\_read\_file\_hex(char \*plainFILE, int \*pbytes):** reads the file 'plainFILE' that contains hexadecimal characters. The function returns the data that was read in bytes.

## 6.2.7 Code Samples

Three examples of library usage are provided in *demo\_block\_ciphers.c*, *demo\_stream\_ciphers.c* and *demo\_hmacs.c*: this program reads the input parameters from the command line and executes one cryptographic function. For block/stream ciphers, the cryptographic function encrypts a plaintext, decrypts the relative cipher text and verifies the decrypted message is the same with the initial plaintext. For *hmacs*, the cryptographic function computes the digest of a message. The program prints relevant error messages. The user can also define input/output files.

For the first demo runs the command: **`$ make block_ciphers`**

The demo compiles all block ciphers. To run the program, type the command:

```
./demo_block_ciphers -o -h <cipher_name> <key_string> <mode> <padding> <input_file>
```

where:

- -o: optional parameter for printing descriptive messages (debugging)
- -h: optional parameter for reading the input data from a file
- <cipher\_name>: AES, DES, TRIPLEDES, PRESENT, LED, KATAN, KTANTAN, CLEFIA, CAMELLIA, XTEA and XXTEA
- <key\_string>: the en/decryption key
- <mode>: ECB, CBC, CTR
- <padding>: zeroPadding, PKCS7, PKCS5, ISO\_10126-2, ISO\_7816-4, X9.23
- <input\_file>: the input file for the '-h' parameter

For the second demo runs the command: **`$ make stream_ciphers`**

The demo compiles all block ciphers. To run the program, type the command:

```
./demo_stream_ciphers -o -h <cipher_name> <key_string> <IV_string> <input_file>
```

where:

- -o: optional parameter for printing descriptive messages (debugging)
- -h: optional parameter for reading the input data from a file
- <cipher\_name>: ARC4, HC128, RABBIT, SALSA20, SOSEMANUK, GRAIN, GRAIN128, TRIVIUM and MICKEY2
- <key\_string>: the en/decryption key
- <iv\_string>: the initialization vector (IV)
- <input\_file>: the input file for the '-h' parameter

For the third demo runs the command: **`$ make hmacs`**

The demo compiles all hmacs. To run the program, type the command:



```
./demo_hmacs -o -h <hmac_name> <key_string> <input_file>
```

where:

- -o: optional parameter for printing descriptive messages (debugging)
- -h: optional parameter for reading the input data from a file
- <cipher\_name>: MD5, SHA-1, SHA-256, SHA-512, SHA-3, KECCAK, BLAKE, JH, GROESTL and SKEIN (SHA-3 actually runs the Keccak function)
- <key\_string>: the digest key
- <input\_file>: the input file for the '-h' parameter

Moreover, we implement a benchmark application (`benchmark_suite.c`) that executes all the compiled primitives over 1 MB data and presents their execution time (the block ciphers are executed in ECB mode with zeroPadding).

After compilation, run the command:

```
./benchmark_suite
```

## 6.3 Anti-tamper technologies

There are basically two kinds of anti-tamper measurements to protect the sensitive information of the node and prevent an easy access by an external attacker [74]:

- Measures that are typically implemented at manufacture level as passive physical barriers.
- Measures consisting of continuous monitoring and detection of tamper attacks. This could be done by means of commercial supervisor chips or through dedicated blocks already implemented in some microprocessors.

Ideally, the safest approach would be to always maintain critical information within a secure chip. Both TPM chips, crypto processors and some other microprocessors already provide some kind of anti-tamper mechanisms and sealed storage. Therefore, depending on the system design, sensitive data flow, execution environment and targeted security level, additional anti-tamper measures may be necessary or not.

It is important to remark that although security level could be increased with better anti-tamper mechanisms, it is not possible to develop a 100% trusted device. The security level of the node must be targeted taking into account its final market. It must be evaluated whether the complexity of the anti-tamper measures compensate the cost of a successful attack or not.

The hypothetical investment that would be required to perform a successful attack would be increased accordingly to the efficiency of the security measurements implemented.

### 6.3.1 Encapsulation and physical barriers

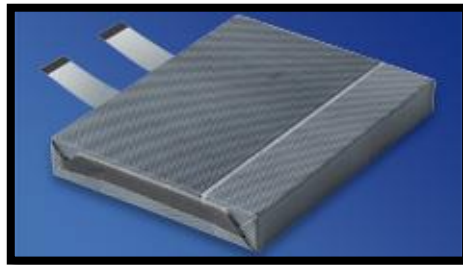
There are some basic guidelines at PCB design phase that could be followed to achieve a minimum protection level:

- Use PCBs with several layers and route the sensitive tracks by intermediate layers only.
- Use BGA-style chips.
- Use blind vias in intermediate layers.
- EMI shielding
- ESD protection

However, these measurements could be circumvented with dedicated tools and instrumentation.

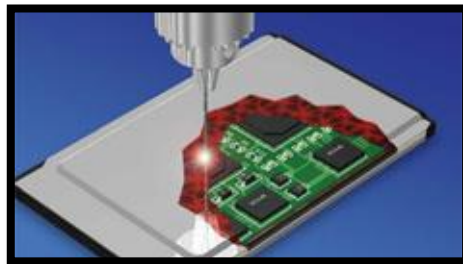
Some manufacturers offer commercial anti-tamper solutions that provide protection for small PCBs by means of dedicated enclosures fully covered with conductive nets. These enclosures contain some monitor points intended to be connected to specialized chips in order to detect physical intrusions (seen as open or short circuits). Some of these products conform to FIPS 140-2 and other security standards:

- GORE: this manufacturer provides some anti-tamper mechanisms based on different enclosure types [75]. The following list is just an excerpt of all available options:
  - Secure encapsulated module: this method is used to provide complete coverage of all surfaces of the PCB. Communication with outside world is done through two flat connectors. It conforms to FIPS 140-2, Level 4.



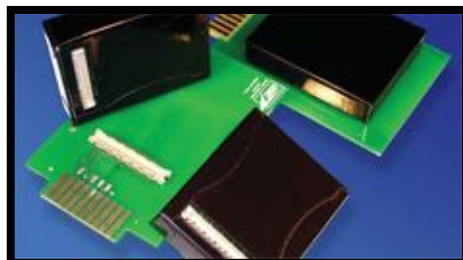
**Figure 6-1 – Secure encapsulated module**

- Secure PCMCIA Card: it is applied to PCMCIA form factor cards. Targeted for FIPS 140-2, Level 4, DoD, NSA Type 1, and CESG Enhanced Grade security



**Figure 6-2 – Secure PCMCIA Card**

- Secure Plug-on Module: the PCB is contained in a rigid secure module which is connected to a motherboard through one single multipurpose connector. This mechanism conforms to FIPS 140-2, Level 3, 3+, and USPS security requirements.



**Figure 6-3 – Secure plug-on module**

- Tamper Respondent Surface Enclosure: consists of an envelope that covers only the critical components of the PCB. It is targeted to meet the requirements of FIPS 140-2, Level 3 and Level 4, DoD, NSA Type 1 security.

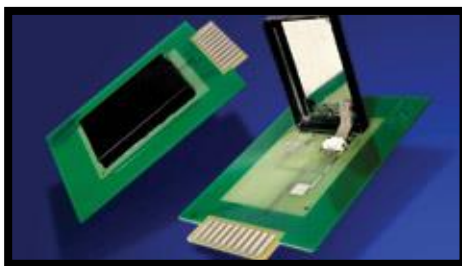


Figure 6-4 – Tamper responsive surface enclosure

- American Energy [76]:
  - Anti-tamper films for electronics

### 6.3.2 Supervisor chips

These chips provide the following monitoring capabilities:

- Detection of short and open circuits (in dedicated nets or circuits).
- Measurement of voltage levels (at critical points).
- Temperature value and temperature change rate.
- Battery leakage.

Therefore, most of these chips usually work in conjunction with some of the encapsulation methods described in 6.3.1, which act as the monitored net/circuit.

Supervisor chips must be continuously powered for right monitoring; usually requiring a current of a few  $\mu\text{A}$ . This involves the utilization of a dedicated battery that limits the lifetime of the device. This battery must be carefully selected regarding aspects such as capacity, self-discharge, temperature range and dimensions.

Examples of this kind of supervisor chips are:

- Maxim [77]:
  - DS36XX family

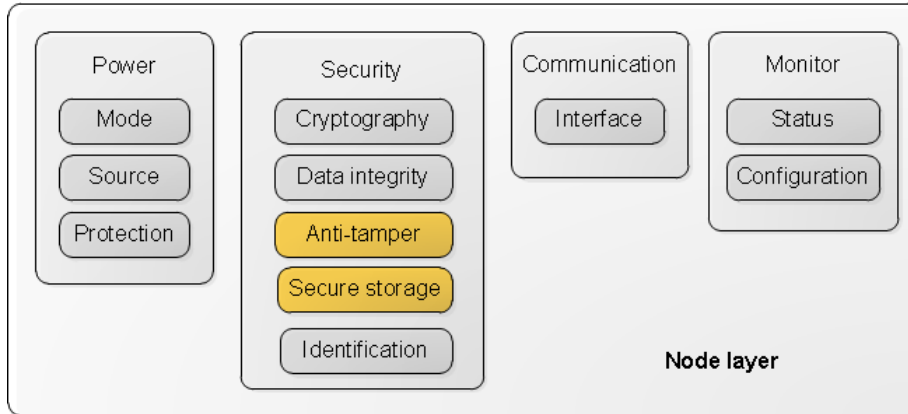
Some of these chips also provide basic secure storage features, such auto-erase small volatile memories.

Usually supervisor chips provide a digital output that change when an attack is detected. The sensitive information should be stored in the volatile memory of the supervisor or any other volatile memory that could be erased if the output is triggered. Therefore the original design of the secured device may be adapted in order to support this kind of chips.

### 6.3.3 Modules interfaces

As commented in previous sections, deliverable D.2.4 presents a proposal of a Reference Architecture for nSHIELD systems.

Technologies described in this section, 6.3, could be used to implement the “*anti-tamper*” component of the *Security Module* and also depend on the anti-tamper technique some attributes of the “*secure storage*” could be implemented, as it is depicted in Figure 6-5.



**Figure 6-5: Reference architecture - Node layer (anti-tamper component)**

Each component should provide some attributes. Next table summarizes the attributes that can be reported by the security module (anti-tamper and secure storage component) with this implementation:

**Table 6-1: Reference architecture – Node Layer (anti-tamper attributes)**

		Attributes	
Security module	Cryptography		
	Data integrity		
	Identification		
	Anti-tamper	Secure key storage	Physical path protection
	Secure storage		

These attributes will be reported to the monitor module or to other component of nSHIELD system. The internal interfaces will be defined.

### 6.3.4 Metrics

The main metrics reported by these components, as defined in deliverable D2.5, could be the metrics related with the *physical / tamper resilience* (“detect” and report the presence of tamper-resistance provisions on the node) and *storage of private information* (boolean value depending on node’s provisions for long-term/fault-condition secure storage of private information).

Taking into account that the current list of metrics is a preliminary one, some other metrics could be added. For example:

- PCB design following recommendations → Boolean value
- Anti-tamper passive (mechanics → resin) → Boolean value
- Security level achieved by the passive anti-tamper (0 – N)
- Anti-tamper active (with supervisor) → Boolean value
- Security level achieved by the active anti-tamper FIPS 140-2

## 6.4 An Identity-Based Encryption scheme

### 6.4.1 Description

Encrypted communications between nodes require a key distribution scheme that will distribute in a secure manner the required keys to the involved nodes. Identity-based cryptography features certain advantages that simplify key distribution and has therefore been preferred. Although the Private Key

Generator (PKG) will have to cope with quite computationally-intensive tasks, the remaining operations can be performed by resource-constrained nodes. Therefore, by assigning the role of the PKG to powerful enough nodes, the use of such a scheme is possible among the heterogeneous nSHIELD network. The Sakai-Ohgishi-KasaharaIBE scheme will be implemented.

#### 6.4.2 Hardware it will be deployed on

The scheme will be initially developed on BeagleBoard-xM and BeagleBone (power and micro nodes, respectively), as they share common characteristics in terms of operating system (Linux) and networking abilities. It will then be extended to other platforms, so as to support more resource-constrained nodes.

Some more detailed hardware specifications are as follows:

- BeagleBoard-xM
  - Operating system: Ubuntu Quantal 12.10 armhf, a version of Ubuntu for ARM processors. An X Window environment is not required.
  - Networking: IPv4, IPv6, IEEE 802.15.4, 6LoWPAN (requires additional hardware). Can connect via its built-in Ethernet port, or through additional hardware attached to its USB port, or its expansion headers.
  - Product website: <http://beagleboard.org/hardware-xm>
- BeagleBone
  - Operating system: Ubuntu Quantal 12.10 armhf, a version of Ubuntu for ARM processors. An X Window environment is not required.
  - Networking: IPv4, IPv6, IEEE 802.15.4, 6LoWPAN (requires additional hardware). Can connect via its built-in Ethernet port, or through additional hardware attached to its USB port, or its expansion headers.
  - Product website: <http://beagleboard.org/bone>
- Crossbow Technology IRIS motes (Nano node)
  - Operating system: Contiki
  - Networking: IEEE 802.15.4, 6LoWPAN
  - Product website: <http://bullseye.xbow.com:81/Products/productdetails.aspx?sid=26>
- Zolertia Z1 (Nano node)
  - Operating system: Contiki
  - Networking: IEEE 802.15.4, 6LoWPAN
  - Product website: <http://www.zolertia.com/ti>

The default operating system of the BeagleBoard-xM and the BeagleBone was Angstrom Linux. However, Ubuntu was chosen because of its wider support community and some networking issues Angstrom had with its networking manager (ConnMan).

The RELIC toolkit [78] supports several cryptographic protocols, including the Sakai-Ohgishi-KasaharaIBE scheme. This scheme will initially be ported to Linux editions that run on BeagleBoard-xM (power node) and BeagleBone platforms (micro node), and some performance statistics will be obtained. As has already been mentioned, the Private Key Generator (PKG) is responsible for generating and distributing the keys to the nodes that wish to communicate between them. However, since this is quite a computationally-demanding operation, the role of the PKG is therefore most suitable to be assigned to power nodes. Certain micro nodes may also be able to support it, but nano nodes should certainly be excluded. Based on the obtained results, the possibility of using a micro node (BeagleBone) as a PKG will be examined. Nano nodes, although they are excluded from becoming PKGs, they should still be able to support the required cryptographic operations, in order to communicate securely with other nodes. This

mechanism will serve as a key exchange scheme that will be incorporated into the IPsec scheme developed for T4.4.

### 6.4.3 Implementation and Security Issues

The RELIC toolkit [78] contains an implementation of the Sakai-Ohgishi-Kasahara IBE protocol. The relevant API includes the functions:

```
void cp_sokaka_gen(bn_t master);
```

Generates a master key for the SOK protocol and stores it in `master`.

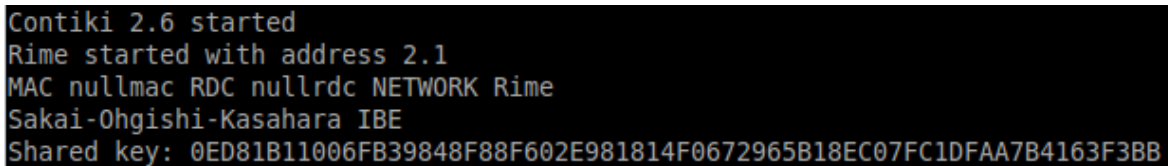
```
void cp_sokaka_gen_prv(sokaka_t k, char *id, int len, bn_t master);
```

Generates a private key (`k`) for identity `id` (having a length of `len` bytes), using the `master` key.

```
void cp_sokaka_key (unsigned char *key, unsigned int key_len, char *id1, int len1, sokaka_t k, char *id2, int len2);
```

Creates a shared key (`key`) between identities `id1` and `id2` (of length `len1` and `len2`, respectively), using the first identity's private key, `k`.

An example screenshot of a key created using the above API on a Linux system is shown in Figure 6-6. The key was created using a 160-bit long prime as the field base, which is able to offer a satisfactory security level, according to the current standards.



```
Contiki 2.6 started
Rime started with address 2.1
MAC nullmac RDC nullrdc NETWORK Rime
Sakai-Ohgishi-Kasahara IBE
Shared key: 0ED81B11006FB39848F88F602E981814F0672965B18EC07FC1DFAA7B4163F3BB
```

Figure 6-6: Key generation using the Sakai-Ohgishi-Kasahara IBE protocol

What also need to be investigated are the memory footprints in both ROM and RAM if the pairings are performed over prime or binary fields, with respect to any potential implications on the execution speed. This will enable making the best possible decision before porting the scheme on nano nodes running the Contiki OS.

## 6.5 Secure Cryptographic Key Exchange using the Controlled Randomness Protocol

### 6.5.1 Introduction

In real world applications of cryptographic protocols, the key management problem refers to the life cycle management of cryptographic keys. It includes the necessary operations for key generation; distribution; storage; replacement and exchange; usage; and destruction. In order to retain specific security level, keys used in cryptographic algorithms and protocols must be periodically refreshed i.e., new keys are exchanged between communicating parties and old keys are replaced.

These precautions ensure that only a specific amount of information is encrypted under the same key and thus, the exposure of information is minimized in case a key is leaked. Key agreement is the process by which two or more parties agree on a common cryptographic key for a specific timeframe. Key transport is the process by which the agreed key is transferred to the participants. In many scenarios, the two

processes occur simultaneously: the participants exchange information by which they both set and exchange the key(s) to be used (or some parts of it).

In many scenarios, the key agreement and transport occur as exchange of control messages through a control channel. This channel does not interfere with the data channel in where actual secure data exchange takes place. A public-key cryptosystem (PKC) is commonly used in such setups in order to securely exchange through the control channel the symmetric-key cryptosystem (SKC) encryption/decryption keys used to securely exchange data within the data channel. The latter keys are often called ephemeral or session keys, since their lifetime spans a specific time period i.e., a session and then they are disposed.

In typical resource-limited environment, like the embedded systems are, it is rather costly to implement and use a public key cryptography (PKC) scheme for secure communication between two entities. When the resource constraints are more severe or the participants are all known beforehand, another option is to replace the “heavy” PKC scheme in the control channel with a lighter SKC scheme. The SKC scheme can use a master key in order to set and transfer the ephemeral keys needed for the data channel. In these cases and for sake of resource economy, the same SKC algorithm can be used in both the “control” and “data” channels albeit with different keys. An embedded system can incur an interesting trade-off on security level and resource consumption.

From a security point of view, the keys must be often refreshed, as explained earlier, in order to maintain the required security level. From a system resource consumption point of view, the keys must be rarely changed, in order to minimize the consumption of precious resources (processor, power, and bandwidth). Further, in some usage scenarios, advanced care must be taken in order to ensure that the new keys will be available by the time they must be used, especially when only intermittent connectivity exists.

The “controlled randomness protocol” (CRP) for cryptographic key management was proposed as an improvement for the security level of secure communication protocols. The CRP allows multiple keys to be valid at any given time; it neither alters the total number of keys needed in the underlying cryptographic algorithms, nor the need of a control channel to periodically refresh keys. However, the increased security offered by CRP allows for far less frequent key exchanges.

The Beagle Board is a low-cost, fan-less single-board computers based on TI's OMAP3 device family, with all of the expandability of today's desktop machines, but without the bulk, expense, or noise. It uses a TI OMAP3530 processor (ARM Cortex-A8 superscalar core ~600 MHz paired with a TMS320C64x+ DSP ~430MHz and an Imagination SGX 2D/3D graphics processor). The Beagle Bone is the latest addition to the BeagleBoard.org family and like its' predecessors, is a low cost ARM Cortex A8 based processor. It has been equipped with a minimum set of features to allow the user to experience the power of the processor and is not intended as a full development platform as many of the features and interfaces supplied by the processor are not accessible from the Beagle Bone via on-board support of some interfaces.

In this chapter we assess the performance of the Controlled Randomness Protocol when implemented on Beagle Board and Beagle Bone embedded systems. We present our findings from two different embedded platforms: one Beagle Board running Embedded Linux and one Beagle Bone running a custom compiled Linux kernel. We provide insights and possible explanations.

The rest of the chapter is organized as follows. Section II presents the key management problem and the controlled randomness protocol. Section III presents our test bed environment and experiments held. Section IV presents the results of our experiments and discusses our findings. Finally, Section V concludes our findings and discusses future directions of the work.

## 6.5.2 The Controlled Randomness Protocol

Conventional cryptographic schemes operate under the assumption that at most one key is active in any time moment. There is only one exception to this assumption. This is the transition periods when changing a cryptographic key. In these cases, at most two keys can be active in order to cope with delayed messages.

We propose a novel approach of having more than one key at any given time moment. The approach is based on the concept of “controlled randomness” i.e., randomly using keys in a controlled environment. The concept of “controlled randomness” can be utilized in any protocol that uses temporal (ephemeral) keys. It increases protocol security with minimal computational overhead. In the following paragraphs we describe the Controlled Randomness Protocol (CRP).

### 6.5.2.1 Protocol Definition

Assume a time period  $t = [0, T]$  composed of time slots  $t_1, t_2 \dots t_n$  such as  $t = t_1 \cup t_2 \cup \dots \cup t_n$ . Each time slot  $t_i$  represents a session. Within each session one specific, temporal cryptographic key  $k_i$  is used in conventional schemes.

The Controlled Randomness Protocol works as follows. Within the time period  $t$  every cryptographic key  $k_1, k_2, \dots, k_n$  is valid and can be used. The sender chooses with a uniform distribution a random integer  $i$  and encrypts the input data using the key  $k_i$ . The receiver has access to a secret mechanism and upon receiving a cipher text  $c_i$  can deduce which of the possible keys was used for the encryption and thus, use the correct one to decrypt the cipher text.

The CRP does not dictate how all these keys are transferred to the receiver. It can be through a control channel using a PKC scheme, or an SKC with master key, or any other method. The CRP dictates how all these keys are used and reused within a time frame composed of many conventional sessions.

Two different methods are originally proposed in for deriving the index,  $j$ , of the secret key used for a given cipher text. The first method is using a synchronized random number generator (RNG) in both the sender and the receiver for the indexes. The second method involves usage of a Keyed Hash Function (KHF) also known as Message Authentication Code (MAC). The sender and the receiver agree on a set of  $n$  encryption keys for a chosen encryption algorithm as usual and additionally on a set of  $n$  keys for computing MAC. The sender further uses an RNG. In this cases, the sender works as follows for every plaintext  $m$ :

- Sender chooses a random number  $j$ .
- Sender encrypts  $m$  under key  $k_j$  to produce the cipher text  $E(m, k_j)$ .
- Sender computes  $H(E(m, k_j), h_j)$  i.e., the MAC of the cipher text using the  $j$ -th MAC key.
- Sender sends  $E(m, k_j) || H(E(m, k_j), h_j)$ , where  $||$  denotes the concatenation operation.

The receiver works as follows to recover  $m$  from the quantity  $E(m, k_j) || H(E(m, k_j), h_j)$ :

- Receiver computes  $H(E(m, k_j), h_j)$  for every possible  $j = 1, 2, \dots, n$ . This step involves at most  $n$  MAC operations. Upon completing all computations, the receiver has derived the secret index  $j$  used by the sender.
- Receiver decrypts  $E(m, k_j)$  using the  $j$ -th decryption key. This step involves one decryption operation and derives the plaintext  $m$ .

### 6.5.2.2 Advantages of CRP

The concept of controlled randomness i.e., having multiple active keys at any given time moment, offers superior security characteristics compared to conventional protocols. The system designer can reuse well-known cryptographic blocks in a novel way to achieve increased security with minimal hassle:

- minimal computational effort can be induced by CRP in the case that both sender and receiver can maintain a synchronized random number generator.
- the synchronization requirement can be relaxed, if the system can sustain some increased computational effort induced by the KHF (MAC) operations.
- in heavily constrained environments, the two above mechanisms can be replaced by sending the random number  $j$  with each packet. In this case, some security is indeed sacrificed since an attacker can know which packet is encrypted under what key. Yet, the intermix of keys allows



consecutive packets to be encrypted under different keys and thus, protect against some cryptanalysis attacks.

The CRP allows in all above scenarios to extend the lifetime of each key way beyond the time of a conventional session.

Further, it allows less frequent exchanges of messages in the control channel (if one is implemented), since less keys are needed to achieve a specific security level for a specific timeframe. An attack on the classical key management protocol with a master key of  $n$  bits has complexity  $O(2^{2n/3})$ ; an attack on the RNG for the controlled randomness protocol with  $m$  keys has complexity  $O(l2m)$  (usually for  $m$ , the period of RNG, it holds  $m \gg n$ ); and an attack on the KHF method has a total complexity of  $O(l(2p + l2_{n/2}))$  where  $p$  the size in bits of the MAC keys.

### 6.5.2.3 Problem Statement

It is argued that the KHF (MAC) method leads to an efficient implementation in the case of combining a symmetric encryption algorithm with KHF operations, since the latter are an order of magnitude faster than the former. This assumption holds if the MAC algorithm is implemented based on a hash algorithm rather than on a symmetric key algorithm. In the simplest scenario, a 0.05% overhead is introduced on average for superior security. In a more complex scenario, this overhead can be lowered to 1%. This is achieved by performing the key detection function every few packets instead of each packet, as in the simpler scenario.

We seek to validate these arguments for overheads in the BeagleBoard family products as a demonstration and stepping stone for the development of the CRP protocol in other embedded systems that are utilized for numerous applications and functionalities.

## 6.5.3 Methodology

### 6.5.3.1 Protocol Implementation in SPD Nodes

We implemented the classical protocols and the two methods of controlled randomness. The protocol implementation simulates a session of data where  $n$  keys are used in total for the three key management variants (classical, CRP with synchronized random number generation, and CRP with keyed hash function).

We used as the underlying symmetric key encryption method the AES algorithm which is readily available in the Linux distribution that we used. On the platform that did not provide readily available AES and HMAC algorithm implementations, we ported those implementations to the Beagle Bone Linux kernel.

### 6.5.3.2 Experiments

We run sets of experiments for release versions of the code. In total, there are four (4) different setups to test. We run two sets of experiments in all platforms. The first set relates to the performance of the CRP depending on the number of keys used.

We run experiments for 4, 8, 16, and 32 keys. We opted to limit the experiments up to 32 keys since it already offers a superior level of security and keeps the average processing overhead relatively low. The second set relates to the performance of the CRP depending on input size. We run experiments for 32, 64, 128, and 256 KB. We opted not to validate for larger sizes given the constrained nature of the device. One should not realistically expect to exchange larger packets with such devices.

## 6.5.4 Results and Discussion

All measurements reported are the average time of 1000 protocol executions. The mean times are in milliseconds. The execution time was measured using the system's tick counts API calls, which offers millisecond granularity. The standard deviation was almost zero in all cases. The execution path remains almost stable in all cases. These sections are still under construction while testing the effects of different cryptographic algorithm implementations on both platforms.

#### 6.5.4.1 Results for Beagle Bone

The experiments ran both on emulator and on a real device. No significant difference was observed in the two sets of measurements. This finding is consistent across all packet sizes and number of keys used. We also verified this in the initial benchmarks we ran for comparing AES with SHA1 implementation. The CRP/RNG implementation offers comparable performance with classical protocols for key management, although the latter do not utilize multiple keys per session. This is an expected behaviour, since, from an implementation point of view, CRP/RNG adds only the use of a random number generator. The CRP/HMAC implementation for the simple scenario (one HMAC per each packet) increases the required computational effort by a linear factor to both the packet size and the number of keys. The increase is between 10% (in case of 4 keys) and 80% (in case of 32 keys). When CRP/HMAC is implemented on the more complex scenario (one HMAC per  $n$  packets), the overheads are lowered, as expected, and within 1-20% range ( $n = 4 : 3.5 - 20\%$ ,  $n = 8 : 2 - 15\%$ , and  $n = 16 : 0.7 - 7\%$ ). We note that it is up to the system designers to decide if, as the number of keys increases, this overhead is acceptable or to opt for fewer keys, without a big sacrifice in the achieved security level. The work for an eavesdropping attacker is smaller but remains exponential to both the symmetric key and the MAC key size.

#### 6.5.4.2 Results for Beagle Board

Initial benchmarking showed that the SHA-1 implementation on .NET CF 4.0 is slower than the SHA1 implementation on the .NET CF 3.5. The faster AES implementation combined with the slower HMAC algorithm results in larger overheads. This is because the performance of the two algorithms does not differ by an order of magnitude any more. We can now see overheads three to six times bigger than before on the same setup. As an example, consider the case of packet size of 64 KB with 32 keys we report a 64%. As long as the results are further verified in a real device, the system designers should use the minimum acceptable number of keys as to minimize the induced computation overhead and retain a high level of security.

### 6.5.5 Conclusions

In this chapter we assessed the performance of the controlled randomness protocol when implemented in the Beagleboard.org family of products. Our findings validate the claims of the initial protocol proposal: the overhead on the encryption side is practically non-existent and only a small, tolerable processing overhead can be induced in the decryption side, despite the increased number of required computations. Thus, the controlled randomness protocol can be a viable implementation option in embedded systems for increasing the actual security of the underlying security protocol. It is an indication that the two platforms incorporate different implementations of cryptography, while sharing a large portion of other code and being akin.

## 7 References

- [1] K. Stammberger, "Current trends in cyber-attacks on mobile and embedded systems," *Embedded Computing Design*, September 2009. [Online]. Available: <http://embedded-computing.com/article-id/?4226>.
- [2] R. Mijat and A. Nightingale, "ARM. Virtualization is coming to a platform near you," 2010-2011. [Online]. Available: <http://www.arm.com/files/pdf/System-MMU-Whitepaper-v8.0.pdf>.
- [3] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," *Commun. ACM*, no. 17, p. 412–421, July 1974.
- [4] VMware, "Understanding Full Virtualization, Paravirtualization, and Hardware Assist," 10 November 2007. [Online]. Available: [http://www.vmware.com/files/pdf/VMware\\_paravirtualization.pdf](http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf).
- [5] "Building a secure system using trustzone technology," ARM Security Technology, 2009. [Online]. Available: [http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C\\_trustzone\\_security\\_whitepaper.pdf](http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf).
- [6] R. Kaiser, M. Conti, S. Orcioni, N. Martínez Madrid and R. Seepold, "Solutions on Embedded Systems," in *Applicability of virtualization to embedded systems*, vol. 81 Lecture Notes in Electrical Engineering, Neatherlands, Springer Netherlands, 2011, pp. 215 - 226.
- [7] J. Hwang, S. B. Suh, S. K. Heo, C. J. Park, J. M. Ryu, S. Y. Park and C. R. Kim, "Xen on ARM: System virtualization using xen hypervisor for arm-based secure mobile phones," in *Consumer Communications and Networking Conference*, Las Vegas, Nevada, 2008.
- [8] G. Klein and R. Kolanski, "Formalising the L4 microkernel API," *CATS '06 Proceedings of the 12th Computing: The Australasian Theroy Symposium*, vol. 51, pp. 53 - 68, 2006.
- [9] C. Gehrmann, H. Douglas and D. K. Nilsson, "Are there good Reasons for Protecting Mobile Phones with Hypervisors?," in *IEEE Consumer Communications and Networking Conference*, Las Vegas, Nevada, USA, 9-12 Jan 2011.
- [10] ARM, "ARM Architecture Reference Manual ARMv7-A," October 2010. [Online]. Available: [http://www.lcs.syr.edu/faculty/yin/teaching/CIS700-sp11/arm\\_architecture\\_reference\\_manual\\_errata\\_markup\\_8\\_0.pdf](http://www.lcs.syr.edu/faculty/yin/teaching/CIS700-sp11/arm_architecture_reference_manual_errata_markup_8_0.pdf).
- [11] "Communications of the ACM:An Interview with Steve Furber," 2011. [Online]. Available: <http://cacm.acm.org/magazines/2011/5/107684-an-interview-with-steve-furber/fulltext>.
- [12] "Open Virtual Platforms," [Online]. Available: <http://www.ovpworld.org>.
- [13] Linux, "Linux Kernel ARM page table," [Online]. Available: <http://lxr.free-electrons.com/source/arch/arm/include/asm/pgtable.h?v=2.6.34;a=arm>.
- [14] Linux, "Linux Kernel ARM memory layout," [Online]. Available: <http://www.kernel.org/doc/Documentation/arm/memory.txt>.
- [15] C. Lambrinoudakis, "Smart card technology for deploying a secure information management framework," *Information Management & Computer Security*, vol. 8, no. 4, p. 173 – 183, 2000.

- [16] M. Drake, "Saving Energy via Smart Power Management," 2012. [Online]. Available: <http://www.ti.com/lit/an/snva630/snva630.pdf>.
- [17] "LTC4365 - UV, OV and Reverse Supply Protection Controller," Linear technology, [Online]. Available: <http://www.linear.com/product/LTC4365>.
- [18] "Supervisory Circuits," Maxim Integrated, 2011. [Online]. Available: <http://www.maximintegrated.com/datasheet/index.mvp/id/5842>.
- [19] T. Instruments, "Power modules, PTH05050W (6 A, 5-V Input Non-Isolated Wide Adjust Module w/ Auto-Track)," 2009. [Online]. Available: <http://www.ti.com/product/pth05050w>.
- [20] M. Turk and A. Pentland, "Eigenfaces for Recognition," *J. Cognitive Neuroscience*, vol. 3, no. 1, pp. 71 - 86, 1991.
- [21] L. Sirovich and M. Kirby, "Low-Dimensional procedure for the characterisation of human faces," *Optical Society of America*, vol. 4, no. 3, pp. 519- 524, 1987.
- [22] M. Kirby y L. Sirovich, «Application of the Karhunen- Loève procedure for the characterisation of human faces,» *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 12, pp. 831-835, 1990.
- [23] D. Liu, K. Lam and L. Shen, "Optimal sampling of Gabor features for face recognition," *Pattern Recognition Letters*, vol. 25, no. 2, pp. 267 - 276, 2004.
- [24] H. Moon and P. J. Phillips, "Computational and performance aspects of PCA-based face-recognition algorithms," *Perception*, vol. 30, no. 3, p. 303–321, 2001.
- [25] D. Liu, K. Lam and L. Shen, "Illumination invariant face recognition," *Pattern Recognition*, vol. 38, no. 10, p. 1705–1716, 2005.
- [26] H. Kyong, I. Chang, K. W. Bowyer and P. J. Flynn, "An evaluation of multi-modal 2d+3d face biometrics," *IEEE Trans. PAMI*, vol. 27, no. 4, pp. 619 - 624, 2005.
- [27] B. Sivakumar Tati, *A framework to implement delegation in offline PACS*, KTH, Sweden: M.Sc. degree thesis, 2012.
- [28] «Keccak - SHA-3 Cryptographic Hash Algorithm Competition,» [En línea]. Available: [http://csrc.nist.gov/groups/ST/hash/sha-3/winner\\_sha-3.html](http://csrc.nist.gov/groups/ST/hash/sha-3/winner_sha-3.html).
- [29] G. Bertoni, J. Daemen, M. Peeters and G. V. Assche, "The Keccak sponge function family," [Online]. Available: <http://keccak.noekeon.org/>.
- [30] C.-Y. Chow, M. F. Mokbel and T. He, "A Privacy-Preserving Location Monitoring System for Wireless Sensor Networks," *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 94-107, 2011.
- [31] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial-of-Service Attacks which Employ IP Source Address Spoofing," Network Working Group, RFC 2827, 2000.
- [32] A. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, B. Schwartz, S. T. Kent and W. T. Strayer, "Single-Packet IP Traceback," *IEEE/ACM Transactions on Networking (TON)*, vol. 10, no. 6, pp. 721 - 734, 2002.
- [33] A. Belenky and N. Ansari, "IP Traceback With Deterministic Packet Marking," *IEEE Communications Letters*, vol. 7, no. 4, pp. 162 - 164, 2003.

- [34] S. Savage, D. Wetherall, A. Karlin and T. Anderson, "Network Support for IP Traceback," *IEEE/ACM Transactions on Networking*, vol. 7, no. 3, pp. 226 - 237, 2001.
- [35] X. D. Song and A. Perrig, "Advanced and Authenticated Marking Schemes for IP traceback," *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, pp. 878-886, 2001.
- [36] K. Stefanidis and D. N. Serpanos, "Packet-Marking Scheme for DDoS Attack Prevention," *Proc. of Security and Protection of Information*, 2005.
- [37] K. Stefanidis and D. N. Serpanos, "Packet Marking Scheme and Deployment Issues," *Proc. of IEEE IDAACS*, 2007.
- [38] "State of IP Spoofing," MIT ANA Spoofer project, 2012. [Online]. Available: <http://spoofer.csail.mit.edu/summary.php>.
- [39] G. Carl, G. Kesidis, R. R. Brooks and S. Rai, "Denial-of-Service Attack-Detection Techniques," *IEEE Internet Computing*, vol. 10, no. 1, pp. 82 - 89, 2006.
- [40] L. Khan, M. Awad and B. Thuraisingham, "A new intrusion detection system using support vector machines and hierarchical clustering," *The International Journal on Very Large Data Bases*, vol. 16, no. 4, pp. 507 - 521, 2007.
- [41] R. R. Kompella, S. Singh and G. Varghese, "On scalable attack detection in the network," *IEEE/ACM Transactions on Networking*, vol. 15, no. 1, pp. 14 - 25, 2007.
- [42] "The Cooperative Association for Internet Data Analysis - CAIDA," [Online]. Available: <http://www.caida.org>.
- [43] W. Feller, *An Introduction to Probability Theory and its Applications*, Vol. 1, vol. 1, John Wiley & Sons (2nd Edition), 1966.
- [44] "The GNU multiple precision arithmetic library," 1991. [Online]. Available: <http://gmplib.org/>.
- [45] "AES implementation," [Online]. Available: <http://www.efgh.com/software/rijndael.htm>.
- [46] "DES implementation," [Online]. Available: <http://www.ubiqx.org/proj/libcifs/source/Auth/>.
- [47] "PRESENT implementation," [Online].
- [48] "The LED Block Cipher," [Online]. Available: <https://sites.google.com/site/ledblockcipher/home-1>.
- [49] "KATAN/KTANTAN implementation," [Online]. Available: <http://www.cs.technion.ac.il/~orrd/KATAN/>.
- [50] "Clefia implementation," [Online]. Available: <http://www.sony.net/Products/cryptography/clefia>.
- [51] "Camellia implementation - a 128-bit block cipher suitable for multiple platforms," [Online]. Available: <https://info.isl.ntt.co.jp/crypt/eng/camellia/>.
- [52] "XTEA - (eXtended TEA) block cipher," [Online]. Available: <http://en.wikipedia.org/wiki/XTEA>.
- [53] "XXTEA - Corrected Block TEA," [Online]. Available: <http://en.wikipedia.org/wiki/XXTEA>.
- [54] "ISO/IEC 7816 part 1-15," [Online]. Available:

[http://www.iso.org/iso/iso\\_catalogue/catalogue\\_tc/catalogue\\_detail.htm?csnumber=29257](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=29257).

- [55] "CyaSSL (implementations of ARC4, MD5, SHA-1, SHA-2)," [Online]. Available: <http://www.yassl.com/yaSSL/Products-cyassl.html>.
- [56] "eSTREAM project: the ECRYPT Stream Cipher Project," 2008. [Online]. Available: <http://www.ecrypt.eu.org/stream/>.
- [57] D. J. Bernstein, "Salsa20 implementation," [Online]. Available: <http://www.ecrypt.eu.org/stream/e2-salsa20.html>.
- [58] "Rabbit implementation," [Online]. Available: <http://www.ecrypt.eu.org/stream/e2-rabbit.html>.
- [59] "HC-128 implementation," [Online]. Available: <http://www.ecrypt.eu.org/stream/e2-hc128.html>.
- [60] "SOSEMANUK implementation," [Online]. Available: <http://www.ecrypt.eu.org/stream/e2-sosemanuk.html>.
- [61] "Grain implementation," [Online]. Available: <http://www.ecrypt.eu.org/stream/e2-grain.html>.
- [62] "Trivium implementation," [Online]. Available: <http://www.ecrypt.eu.org/stream/e2-trivium.html>.
- [63] "MICKEY v2 implementation," [Online]. Available: <http://www.ecrypt.eu.org/stream/e2-mickey.html>.
- [64] "SHA-3 / Keccak implementation," [Online]. Available: <http://keccak.noekeon.org/>.
- [65] "SHA-3 contest," [Online]. Available: [http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/submissions\\_rnd3.html](http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/submissions_rnd3.html).
- [66] "Blake implementation," [Online]. Available: <https://131002.net/blake/>.
- [67] "JH implementation," [Online]. Available: <http://www3.ntu.edu.sg/home/wuhj/research/jh/>.
- [68] "Groestl implementation," [Online]. Available: <http://www.groestl.info/>.
- [69] "Skein implementation," [Online]. Available: <http://www.skein-hash.info/>.
- [70] "Memsic Iris Manual," [Online]. Available: [http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS\\_Datasheet.pdf](http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf).
- [71] "BeagleBone Manual," BeagleBoard.org, [Online]. Available: <http://beagleboard.org/bone>.
- [72] "BeagleBoard Manual," BeagleBoard.org, [Online]. Available: <http://beagleboard.org/hardware>.
- [73] "BeagleBoard-xM Manual," BeagleBoard.org, [Online]. Available: <http://beagleboard.org/hardware-xm>.
- [74] "Embedded Security," Maxim Integrated, [Online]. Available: <http://www.maximintegrated.com/products/embedded-security/>.
- [75] "Anti-Tamper Physical Security for Electronic Hardware," GORE, [Online]. Available: [http://www.gore.com/en\\_xx/products/electronic/anti-tamper/anti-tamper-respondent.html](http://www.gore.com/en_xx/products/electronic/anti-tamper/anti-tamper-respondent.html).

- [76] "Revolutionary Technologies: Armor on flexible substrates and Anti-Tamper Coatings," American Energy Technologies Company, [Online]. Available: <http://usaenergytech.com/armor-and-anti-tamper-coatings-on-flexible-substrates/>.
- [77] "Supervisors, Voltage Monitors, Sequencers," Maxim Integrated, [Online]. Available: <http://www.maximintegrated.com/products/supervisors/>.
- [78] "An efficient library for cryptography," Relic Toolkit, [Online]. Available: <http://code.google.com/p/relic-toolkit/>.