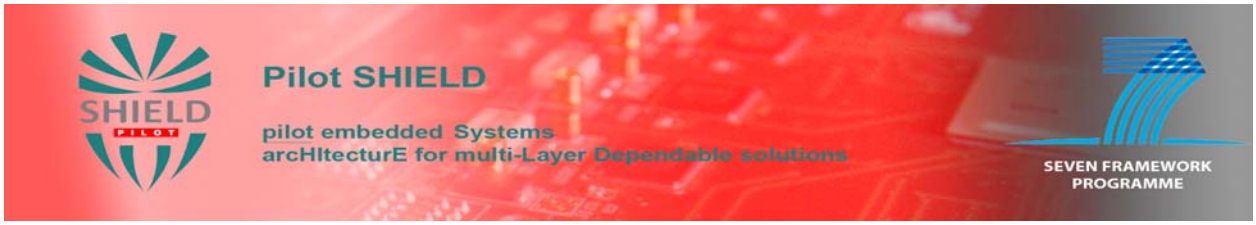

Project no: 100204

pSHIELD

pilot embedded **S**ystems arc**HI**tectur**E** for multi-**L**ayer **D**ependable solutions

Instrument type: Capability Project

Priority name: Embedded Systems / Rail Transportation Scenarios

SPD node technologies prototype

For the pSHIELD-project

Deliverables D3.1

Partners contributed to the work:

SESM, Italy
ETH, Italy
CS, Portugal,
ATHENA, Greece,
AS, Spain,
CWIN, Norway,
MAS, Norway

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

Document Authors and Approvals			
Authors		**Date**	**Signature**
Name	**Company**		
Przemyslaw Osocha	SESM		
João Cunha	SESM		
Fabio Giovagnini	SESM		
Jose Verissimo	CS		
Josef Noll	MAS		
Silvia Mier	AS		
Paolo Azzoni	ETH		
Mohammad Chowdhury	CWIN		
Kyriakos Stefanidis	ATHENA		

Modification History		
Issue	**Date**	**Description**
Draft A	24 May 2011	First ToC proposal for comments
Draft B	9 September 2011	Incorporates comments from Draft A review
Issue 1	15 September 2011	Incorporates comments from Draft B review
Issue 2		Incorporates comments from issue 1 review

Contents

Figures

Tables

Glossary

ESs	Embedded Systems
SPD	Security Privacy Dependability
UCS	Use case Scenario
HW	Hardware
SW	Software

This Page is intentionally left blank

Executive Summary

Deliverable D3.1 "SPD node technologies prototype" is document comprising output from all tasks of work package WP3 "SPD Node", so Task 3.1 "Nano, Micro/Personal node", Task 3.2 "Power node" and Task 3.3 "Dependable self-x and cryptographic technologies". This document is kind of introduction to the final prototype reports, which will be output of every WP3 task, respectively: D3.2 "SPD nano, micro/personal node technologies prototype report", D3.3 "SPD power node technologies prototype report" and D3.4 "SPD self-x and cryptographic technologies prototype report".

The structure of D3.1 is divided into three main chapters corresponding to three tasks of WP. Every task contributes to appropriate part of this deliverable.

General objectives of WP3 are as follow:

- Select a representative set of SPD technologies at Node level;
- Develop appropriate composability mechanisms at such level;
- Deliver a SPD node prototype.

WP3 plays important role in designed four layers pSHIELD architecture, representing the basic components of the lower part of the SPD Pervasive System: Node Layer.

Work package 3 works interact with other project tasks, e.g. contribution coming from research and development performed in WP2 and WP4, which are strictly interconnected and interdependent with WP3 and results will be used in WP6. Task 2.1 will provide the requirements and specification for a prototype. Task 2.3 will provide definitions of proper interfaces that will allow the nano, micro/personal nodes interoperation with the rest of SHIELD platform. WP4 provides Task 3.3 with SPD features at network level to be implemented at node level.

The aim of deliverable D3.1 is to introduce solutions selected in WP3 to fulfill work package goals. The full description of mentioned solutions will be presented in following up WP3 deliverables.

1 Introduction

Work Package 3 covers problematic of 3 different kinds of Intelligent ES Nodes: nano node, micro/personal node and power node. These three node types represent the basic components of pSHIELD architecture, creating Node Layer, one of four layers, beside Network, Middleware and Overlay Layers.

The WP aims at providing SPD intrinsic capabilities at node layer through the creation of an Intelligent ES HW/SW Platform consisting of three different kinds of Intelligent ES Nodes: nano node, micro/personal node and power node. These three node types (which can be considered three node levels of increasing complexity) will represent the basic components of the lower part of the SPD Pervasive System, and 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.

Objectives of Work package 3 "SPD Node" are: selection of a representation of SPD technologies at Node level, development of appropriate composability mechanisms at node level, and deliver a SPD node prototype.

Aim of this deliverables D3.1 is to present SPD node technologies prototype. Prototypes of such SPD technologies are developed, following the composability criteria of the pSHIELD architecture design delivered by WP2.

Each WP task contributes to improve and enable the composability mechanisms of the whole node-specific set of SPD technologies. Nonetheless each task will affect and give specific attention to the development of specific technologies.

Nodes definitions

pSHIELD SPD Architecture is composed of four layers:

- Node Layer,
- Network Layer,
- Middleware Layer,
- Overlay Layer.

Node Layer represents the basic components of the lower part of the SPD Pervasive System.

That layer consisting of three different kinds of ES Nodes which can be considered three node levels of increasing complexity:

- Nano Node,
- Micro/Personal Node,
- Power Node.

Nano Node level typically consists of small, mainly wireless sensors, with limited HW and SW resources. Because of their massive distribution in the environment, they could become an interesting target for attacks and hacking.

Micro/Personal Node level consists of devices richer than the Nano Nodes in terms of hardware and software resources, network access capabilities, mobility, interfaces, sensing capabilities etc. The specific functions of a Micro/Personal Node are generally referred to:

- secure network access capabilities,
- monitoring and sensing,
- interfacing.

Power Node level represents, in the pervasive system, the first level of massive data elaboration, with the peculiarity that the computing power is provided directly on the field.

Our definition of nodes follows a classification provided in following table, indicating hardware, software, networking capability, mobility, interfaces and sensing capabilities as criteria on how to classify the nodes.

Table 1 – Characteristics of pSHIELD nodes

	nano	micro	personal	power
hardware	restricted		extendable, personal/mobile board,	typical data center, server board
software	not changeable (sense and send)	process (collect and store)	decision making, (java)	ontologies, reasoning
network access	through gateway		direct wireless/mobile	fixed (backend)
mobility	-		yes	no
interfaces	fixed or wireless		network and usb-like	fixed
sensing	one or more sensors, e.g. light, temperature, position		inbuild sensors	no

Reason for such a classification is the ability on introducing SPD functionality on those sensors. The following examples of sensors are used to explain the capability for SPD extensions. Examples of sensors in the identified categories are:

- Power node: Datasenter for ontologies and reasoning
- Personal node: mobile phone, embedded linux system
- Micro-node: SunSpot
- Nano: GPS unit

The following short introduction on capabilities should be seen as an example, in-depth discussions are foreseen in the deliverables D3.2 and D3.3.

A GPS unit, attached by cable to an USB port of a personal or power node, can be seen as an example of a nano node. As the hardware is sealed, the only SPD functionality would be the monitoring of the attached hardware address through the USB port in order to avoid a replacement with another unit. No other operations can be performed with the sensor itself. Communication in this example is through the USB wire, but other nano nodes like thermometers or vibration units might work on a wireless interface, typically ZigBee-, Bluetooth- or WLAN-based. What is in common is that network attachment can be controlled, but that the communication protocol typically supports only standardized exchanges.

Micro-nodes like a SunSPOT sensor have a limited capability to process information, to register sensor units and to communicate to the surrounding. As their hard- and software is limited, modifications to ensure SPD functionality are tedious, if not impossible. This is one of the reasons for the development of own hardware in pSHIELD.

A mobile phone is a typical example of a personal node. The device itself has the capability of having SPD-related functionalities, such as ID control, and adjustments to communication protocols. Typical

embedded software based on Linux can also be adjusted to fulfill the needs for increased security. As processing capability is available to quite some extend, operations on information is possible. However, not all available software libraries are supported, and some internal libraries might be blocked for user access. Thus application support will typically be based on access control and decision making structures. More complex operations such as reasoning and ontology middleware support are often not possible on personal nodes.

Power nodes are seen as the back-end of an embedded installation, typically a server which may control access and admin of the embedded systems. These back-end systems will enable to perform the reasoning based on the sensor data from the embedded sensors, and will also be able to delegate decision-making to underlying personal nodes.

The following sections will provide an overview over the envisaged SPD functionalities on the embedded nodes.

2 Nano, Micro/Personal node SPD technologies prototype [MAS, CWIN]

This document tries to highlight the work areas of the consortium members within the scope of this project. We have identified the following areas for SPD related work in pSHIELD:

- Secure firmware developments
- Implementation of security across heterogeneous platforms.

CWIN is prototyping a demonstrator consisting of nano-, micro- and personal nodes, the document also indicates the integration aspects of this demonstrator. The platform currently consists of two different personal nodes, a mobile phone and an embedded Linux platform. The embedded Linux platform connects to a GPS nano-sensor through a wired connection (USB), and has a wireless connection to a Sun SPOT device being able to measure accelerometer, temperature, and light.

The implementation uses the Sun SPOT sensor platform as micro node. Sun SPOT is a useful platform for developing and prototyping application for sensor network and embedded system. Sun SPOT is suitable for application areas such as robotics, surveillance and tracking. The main units are Sunspot devices with embedded sensors and base station. Each Sunspot has a so-called eSPOT with battery, while the base station is not equipped with battery and must be powered from the host computer via an USB cable. The Sunspot does not need to run any operating system, it needs only JVM that runs on bare metal, and executes directly out of ash memory.Stack-boards composed of specific sensors and actuators such as accelerometers, light sensor and temperature sensor.

The integrated sensors include as integrated Sensors:

- Temperature Sensor: Chip-type is ADT7411 sensor that measures temperature with ADC. ADC is integrated into the Demo, and can measure temperatures between -40°C to +125°C.

- Accelerometer sensor: 3-axis accelerometer of the type LIS3L02AQ, designed by ST Micro Systems and is in eDemo Board. This sensor can measure the x-axis, y-axis and z-axis in the direction up and down with the value either ±2G or ±6G. When the Sunspot is at rest, it measures x = y = 0 and z = 1G.

- Light sensor is of the type TPS851, designed by Toshiba. The sensor can measure the voltage between 0.1V (dark) - 4.3V (light), and converts the voltage to the brightness of Luminance (lx) 3.

1.1 These sensors are controlled through a VIA EPIA N700 board, which is a compact, low heat, power-efficient Nano-ITX board, ideal for compact industrial PCs and embedded automation devices. The board is integrated with the VIA VX800 media system processor, an all-in-one chipset solution that provides an extensive feature set while using less real state, helps to make the VIA EPIA N700 a superb choice for compact systems. It is based on Nano-ITX form factor (12cm x 12cm. VGA, USB, COM, Compact Flash (CF) and Gigabit network ports are provided on the board to help reduce system foot-print size and eradicate cluttered cabling for improved air-flow and enhanced stability in always-on systems. The VIA VX800 offers an integrated DirectX9 graphics core and excellent hardware accelerated video playback for MPEG-2, WMV9, VC1 video formats. An on-board VGA port is provided along with support for DVI and a multi-configuration 24-bit, dual channel LVDS transmitter, enabling display connection to embedded panels. The VIA EPIA N700 is equipped with a power-efficient 1.5GHz VIA C7, supports up to 2GB of DDR2 system memory and includes two onboard S-ATA connectors, USB 2.0, COM and Gigabit LAN ports. Expansion includes a Mini-PCI slot with an IDE port, additional COM and USB ports and PS/2 support available through pin-headers. The VIA EPIA

N700 offers total system stability at extreme temperatures ranging from -20°C to 70°C, an ideal solution for our Norwegian rail use case to meet the extreme weather condition of Norway.

2.1 Potential SPD functionalities in the demonstrator [MAS, CWIN]

So far effort was given to integration through the third party an open platform of Telenor Objects. This integration has only limited considerations on security, privacy and dependability (SPD). However the communication with the Telenor Shepherd platform is secured using HTTPS protocol. The following scenarios can illustrate how the above implementation can be extended with the SPD aspects. The first two scenarios address the security aspect and the third scenario deals with the dependability situation.

Scenario 1: Only valid node can be connected to the system (illustrated in fig. 6). In this regard, we may use mobile phone as personal node and can only be integrated with the system if it is authenticated by the system. This will avoid getting communication and data from a fake node.

Scenario 2: Communication to 'new sensor' is only allowed if it is taking place at a pre-defined location. In this regard we assume that the location is a restricted place and only authorised user can access the premise. The system will get the location information from GPS sensor and if the location is same as the 'pre-defined' location, the system can communicate with the 'new sensor' installed on-board. Scenarios 2 can further secure the new sensor integration on top of scenario 1.

Scenario 3: As GPS reception is typically very poor inside the measurement vehicle, we needed to address location through a combined approach of two independent sensor. We extended the set-up with a mobile phone, which allows us position based on three methods: GPS, network and WLAN coverage. Taking into consideration the time-stamp of the equipment will open for a "dependable" positioning solution through a composition of sensor data from the embedded linux platform and the mobile phone.

The realization of the scenarios is subject to further discussion.

2.2 Integration towards Telecom systems [MAS, CWIN]

The main focus of the work in this area has been on the integration of the combined sensor platform into the telecom infrastructure of Telenor, the *Shepherd*® Platform.

Telenor, Norway have introduced a platform (named as Shepherd®) for interoperability and integration that supports communication between connected devices (nano and micro nodes) and makes them accessible from anywhere at anytime.

The Shepherd® is a platform for Connected Objects (COs) [32]. This means that any the pluggable component can be connected, and be integrated in Shepherd® platform as a Connected object (CO). Figure below depicts the overview of Shepherd® platform.

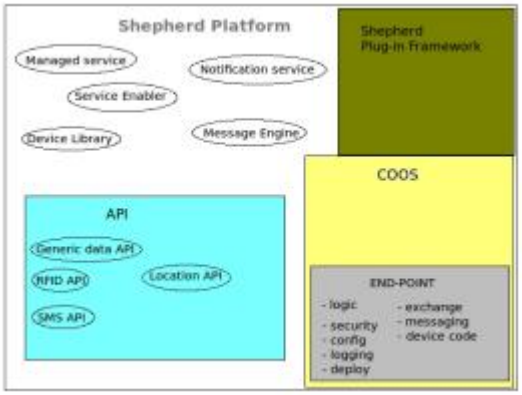

Figure 1 – Details of Telenor's Shepherd platform

The platform offers number of service including:

- Service Management for monitoring, device configuration, SLAs, and supporting.
- Service Enabler has a specific API that allows further access to other modules.
- Message Engine handles and secures the process of message flow, including capturing, processing, routing and storage of data in an environment.
- Notification services that inform about the status of devices and applications.
- Device library consists of interfaces for tools and services recognition.

Connectivity with Shepherd® Platform Shepherd® offers two methods for establishing connection. This includes: 1. HTTP Connection API - This mechanism establishes a direct connection to the Shepherd by using the HTTPS protocol. With this method, it requires the development of the HTTP API of the object. Shepherd accepts both methods POST and GET. When the connection is established, the Shepherd sends a response code back to that object as a confirmation of success or failure of reception. To be able to connect to the Shepherd, the "device object" is identified with an Application ID and an Object ID.

2. Connected Objects Operating System (COOS) - is an open source and has been written in Java. When using the COOS instance, the applications can connect to Shepherd in a secure, reliable and stable manner. In particular, it is important in this respect that eavesdropping by third parties is not possible when using COOS. Reliable in the sense that it is an M2M network, and communication between objects with COOS instance and the Shepherd will not be interrupted or delayed more than necessary. Thereby, ensuring a stable environment for the users and the applications. It requires therefore, developing an application using COOS, so this can apply to device so it can communicate with the Shepherd. From a programming perspective, "Connected Objects

Operating System (COOS) is an application distributed in a container so that it can enable data exchange between the object and the Shepherd. In COOS concept, every component that is integrated, and can be pluggable is called "Connected Object (CO). This means that a COOS instance can have multiple Connected Object, and each COOS instance carries its own distinctive character.

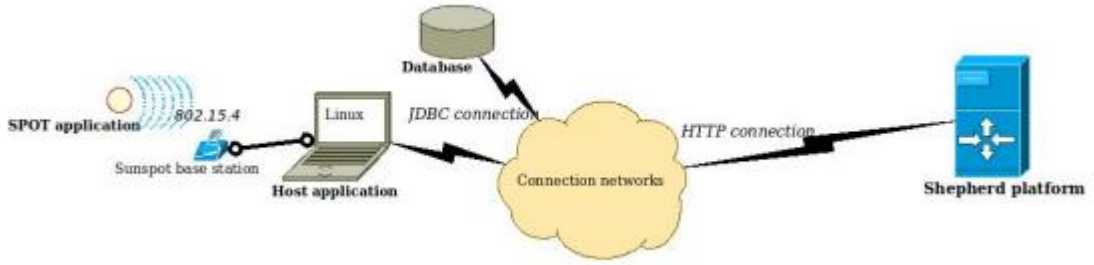

Figure 2 – Set-up of the communication from SunSPOT to the Shepherd platform

Figure given above presents the system overview. It shows the establishment of an intended two-way communication between Sun SPOT sensors and its base station, and also two-way communication between the embedded Linux system and the Shepherd Platform.

A Host application has been developed, that performs broadcasts every fifteen seconds. While, the spot application will detect the broadcasts every thirty seconds. But, it does not transmit the values to the base station after one minute has passed since the last envoy. When the values arrive, these will also be stored in cache. At the same time, the Host application sends out a request to Shepherd for receiving the values. The connection is opened until the application has received confirmation from the Shepherd of receipt. However, the values to be sent to Shepherd, only happens in every five minutes. The SPOT application is also designed to detect spot's battery level prior it using the wireless communication. If spot battery is either lower than -32 or greater than 32, then the MIDlet will be destroyed, terminated and a notification will be send to the concerned actors.

3 Power Node SPD technologies prototype [ETH, SESM, AS, CWIN]

3.1 SPD Power Node Layer prototype [SESM]

In many industrial, telecommunications or transportation appliances the transmission of digital information through noisy environments makes use of Frequency-shift keying (FSK) due to its immunity to "adverse environment" conditions (i.e. electromagnetic interference, noise, surge, ground loop/ground plane shift problems), its ability to transmit data across commutators or sparking sources (sliding contacts, slip rings, rolling wheels, etc.), and the use of any two conductor wire, shielded or unshielded. Furthermore, it is employed even in wireless communications, such as in digital cellular communications system (GSM), using Gaussian minimum shift keying (GMSK), a special type of FSK.

This case study proposes the use uses FSK modulation to transmit data between intrusion sensors placed in different cars of a freight train, into an SPD Power Node. The sensors must be equipped with a modulator, and each transmit using different carriers. The Power Node receives the signals, demodulates them, processes the data, encrypts it and sends to a control center through the pSHIELD Network.

3.1.1 Power Node use-case scenario

This scenario demonstrates the Node Layer capabilities, but some Network, Middleware or Overlay functionalities may also be used.

This case study is composed of:

- A single proximity sensor, emulated by an FPGA based board, continuously simulating the distance to the closest object.
- A data encryptor, running in the same board, encrypting this data.
- A FSK modulator, modulating the encrypted data into a FSK signal. This is also performed by the same FPGA based board
- A parallel 8 bit wide data bus with the synchronization clock line between the signal generator and the Power Node.
- A SPD Power Node, built with a board with a Xilinx FPGA. This Node has a FSK demodulator, a data decryptor, and a web server, presenting the node status, metrics and received data.
- A fault-injector, activated by a pushbutton, able to inject a fault into the FPGA.
- A Control Center, which is a PC with a web browser.
- An Ethernet connection between the SPD Node and the Control Center.

The following scenarios shall be demonstrated:

- The PC continuously presents some data on the screen, simulating a distance, modulates this data, and transmits it to the SPD Node. The SPD Node demodulates the signal, and exposes its data in the web browser. The Control Center PC shows this web page with the same simulated distance.

- While the scenario 1. is executing, a fault is injected into the demodulator. An error is detected and recovered, by a FPGA reconfiguration. Correct data is still presented to the Control Center. The metrics reveal that an error has occurred, and recovery was successful.
- The Modulator switches to a different carrier. The SPD Node detects this error, and the demodulator is automatically reconfigured to this new carrier. On the Control Center, data is still valid. The status reveals a new carrier is being used.

3.1.2 SPD Capabilities Demonstration

This prototype shall demonstrate the following SPD capabilities and functionalities:

- Legacy component adaptation to pSHIELD, by providing SPD functionalities to a legacy FSK Demodulator.
- Dependability, by detecting errors in the demodulator, and tolerating them, through FPGA reprogramming.
- Security, by receiving encrypted data and being able to decrypt it.
- Self-Reconfiguration, by detecting that a different carrier is being used in the FSK signal, and reconfiguring the FPGA for the new carrier.
- Metrics, by collecting and providing data such as the number of messages received, errors detected, etc.
- Composability, by providing discovery and composability information, such as the identification of the modules and its characteristics, that build-up the SPD Node.
- High performance, by demodulating and decrypting in real-time all the received information.

3.1.3 A-FSK Demodulator Context

The context application, used to demonstrate how the innovative “SDP Node” could be compliant with security, dependability and privacy requirements of pSHIELD Project is showed in the following figure.

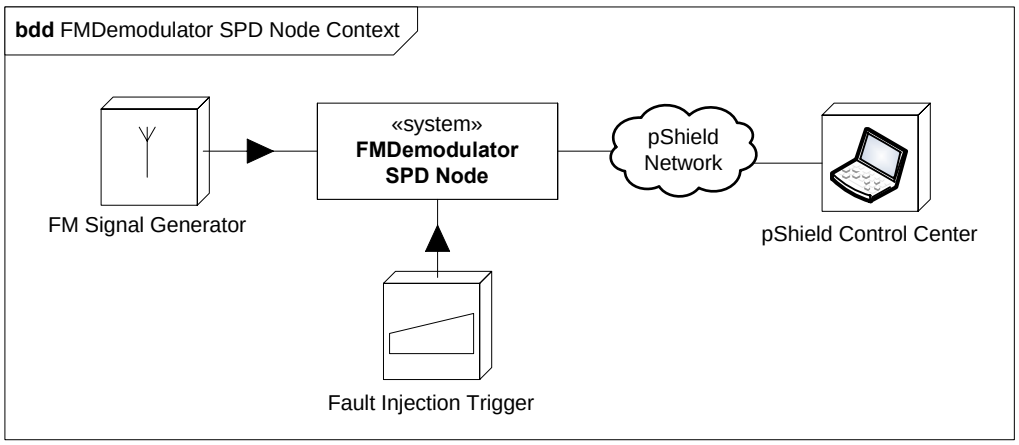

Figure 3 – A-FSK Demodulator demonstration context

The context application refers to an "**A-FSK Demodulator SPD Node**" which has been implemented and developed according to the general architecture that defines the features of a SPD Node device.

3.1.4 A-FSK Demodulator Node Layer Description

The A-FSK Demodulator is a proof of concept to demonstrate the SPD paradigm. In fact it implements a simple system managing a data stream with the constraints and suggestions of the SPD paradigm.

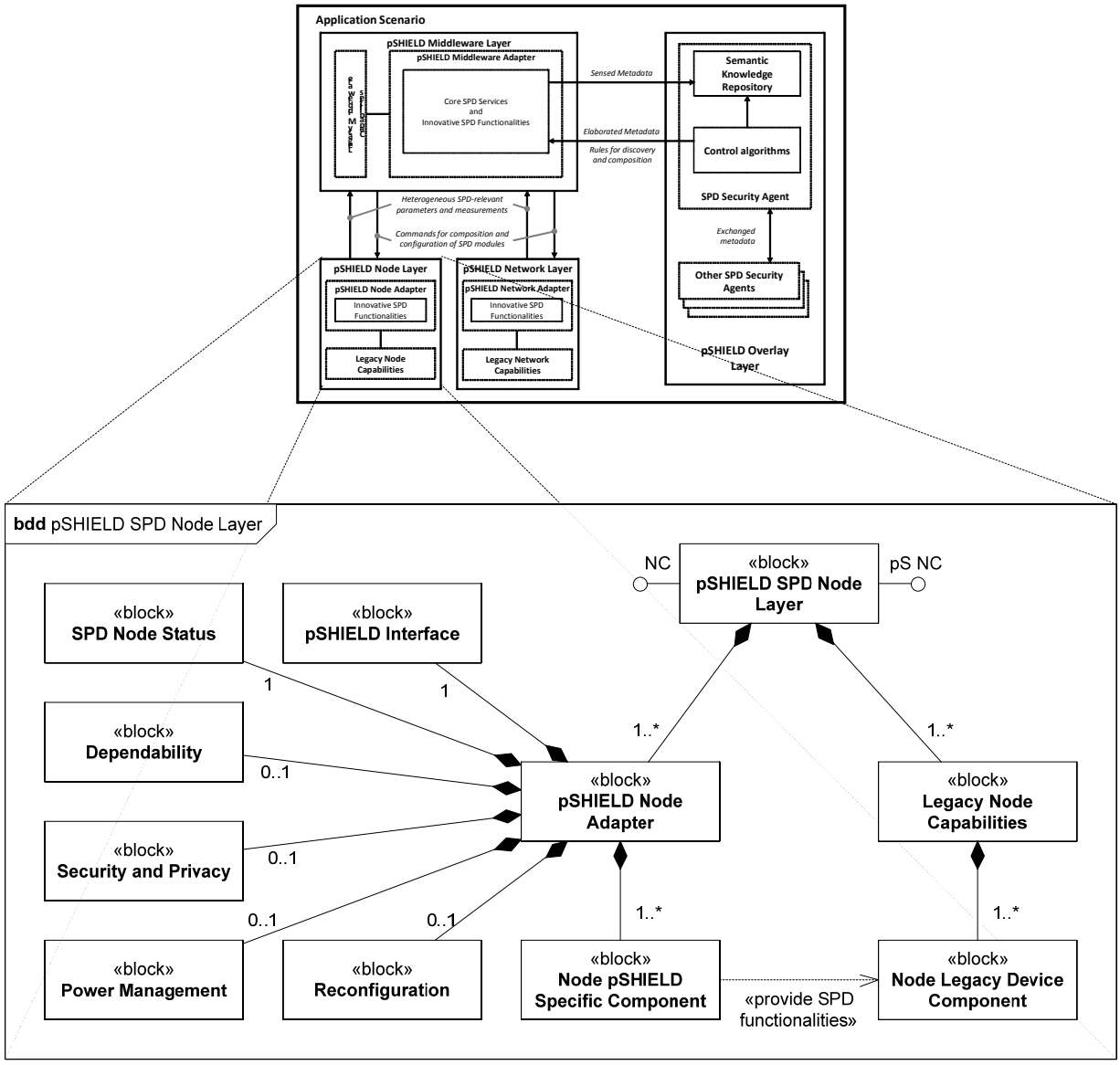

Figure 4 – Generic pSHIELD SPD Node Layer

The demonstrator implements the following features:

- a reconfigurable demodulator,
- a fault injection mechanism,
- a secure connection with the network layer.

The reconfigurable module allows the system to be dependable, and the fault injection mechanism shows how the system can resume its capabilities after a fault.

The secure connection allows the node to be reliable for the network layer and for the important information exchanging and application running at such a level.

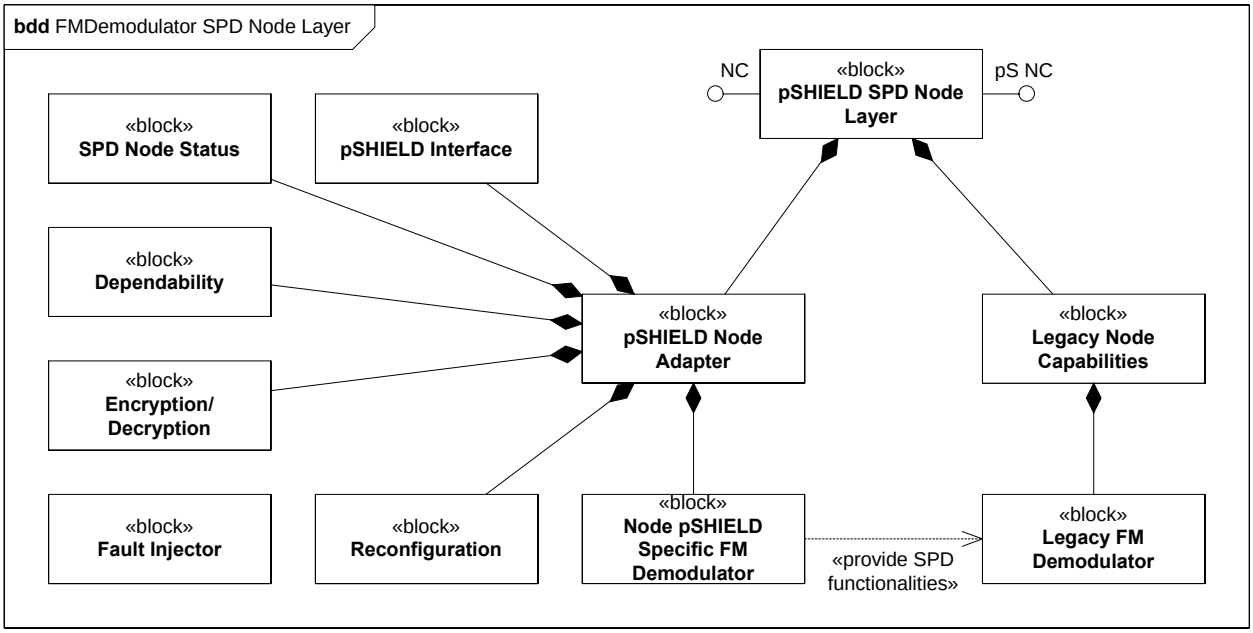

Figure 5 – A-FSK Demodulator SPD Node Layer

3.1.5 System Architecture

The following figure shows the block diagram of the hardware implementation of the A-FSK Demodulator SPD Node.

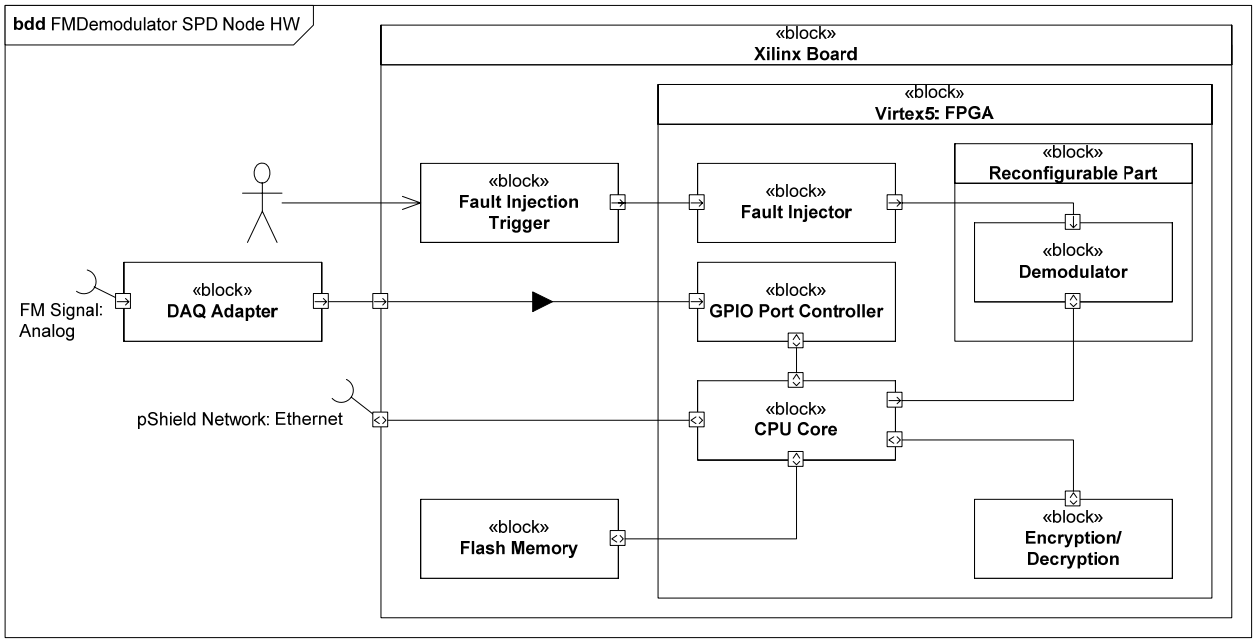

Figure 6 – A-FSK Demodulator hardware architecture

For our purposes, it has been used a **Xilinx ML507 Evaluation Board,** technical specification is given below.

Features:

- Xilinx Virtex-5 FPGA
  - XC5VFX70T-1FFG1136 (ML507)
- Two Xilinx XCF32P Platform Flash PROMs (32 Mb each) for storing large device configurations
- Xilinx System ACE™ CompactFlash configuration controller with Type I CompactFlash connector
- Xilinx XC95144XL CPLD for glue logic
- 64-bit wide, 256-MB DDR2 small outline DIMM (SODIMM), compatible with EDK supported IP and software drivers
- Clocking
  - Programmable system clock generator chip
  - One open 3.3V clock oscillator socket
  - External clocking via SMAs (two differential pairs)
- General purpose DIP switches (8), LEDs (8), pushbuttons, and rotary encoder
- Expansion header with 32 single-ended I/O, 16 LVDS-capable differential pairs, 14 spare I/Os shared with buttons and LEDs, power, JTAG chain expansion capability, and IIC bus expansion
- Stereo AC97 audio codec with line-in, line-out, 50-mW headphone, microphone-in jacks, SPDIF digital audio jacks, and piezo audio transducer
- RS-232 serial port, DB9 and header for second serial port
- 16-character x 2-line LCD display
- One 8-Kb IIC EEPROM and other IIC capable devices
- PS/2 mouse and keyboard connectors
- Video input/output
  - Video input (VGA)
  - Video output DVI connector (VGA supported with included adapter)
- ZBT synchronous SRAM, 9 Mb on 32-bit data bus with four parity bits
- Intel P30 StrataFlash linear flash chip (32 MB)
- Serial Peripheral Interface (SPI) flash (2 MB)
- 10/100/1000 tri-speed Ethernet PHY transceiver and RJ-45 with support for MII, GMII, RGMII, and SGMII Ethernet PHY interfaces
- USB interface chip with host and peripheral ports

- Rechargeable lithium battery to hold FPGA encryption keys
- JTAG configuration port for use with Parallel Cable III, Parallel Cable IV, or Platform USB download cable
- Onboard power supplies for all necessary voltages
- Temperature and voltage monitoring chip with fan controller
- 5V @ 6A AC adapter
- Power indicator LED
- MII, GMII, RGMII, and SGMII Ethernet PHY Interfaces
- GTP/GTX: SFP (1000Base-X)
- GTP/GTX: SMA (RX and TX Differential Pairs)
- GTP/GTX: SGMII
- GTP/GTX: PCI Express® (PCIe™) edge connector (x1 Endpoint)
- GTP/GTX: SATA (dual host connections) with loopback cable
- GTP/GTX: Clock synthesis ICs
- Mictor trace port
- BDM debug port
- Soft touch port
- System monitor

Top view of the ML507 printed board assembly is shown in the figure below.

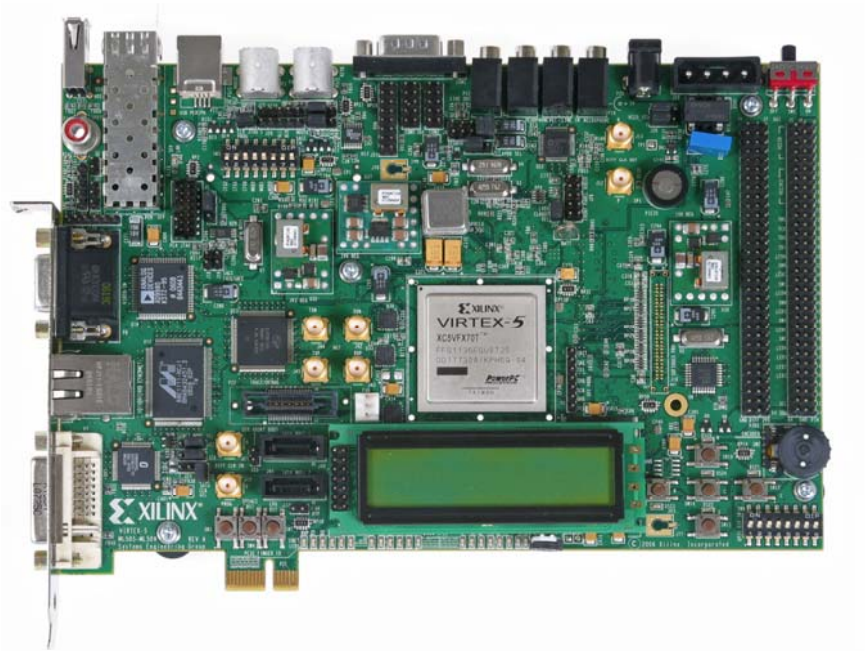

Figure 7 – ML507 printed board assembly (top view)

Block Diagram of Xilinx ML507 Evaluation Platform used for the demonstrator is shown in the next figure.

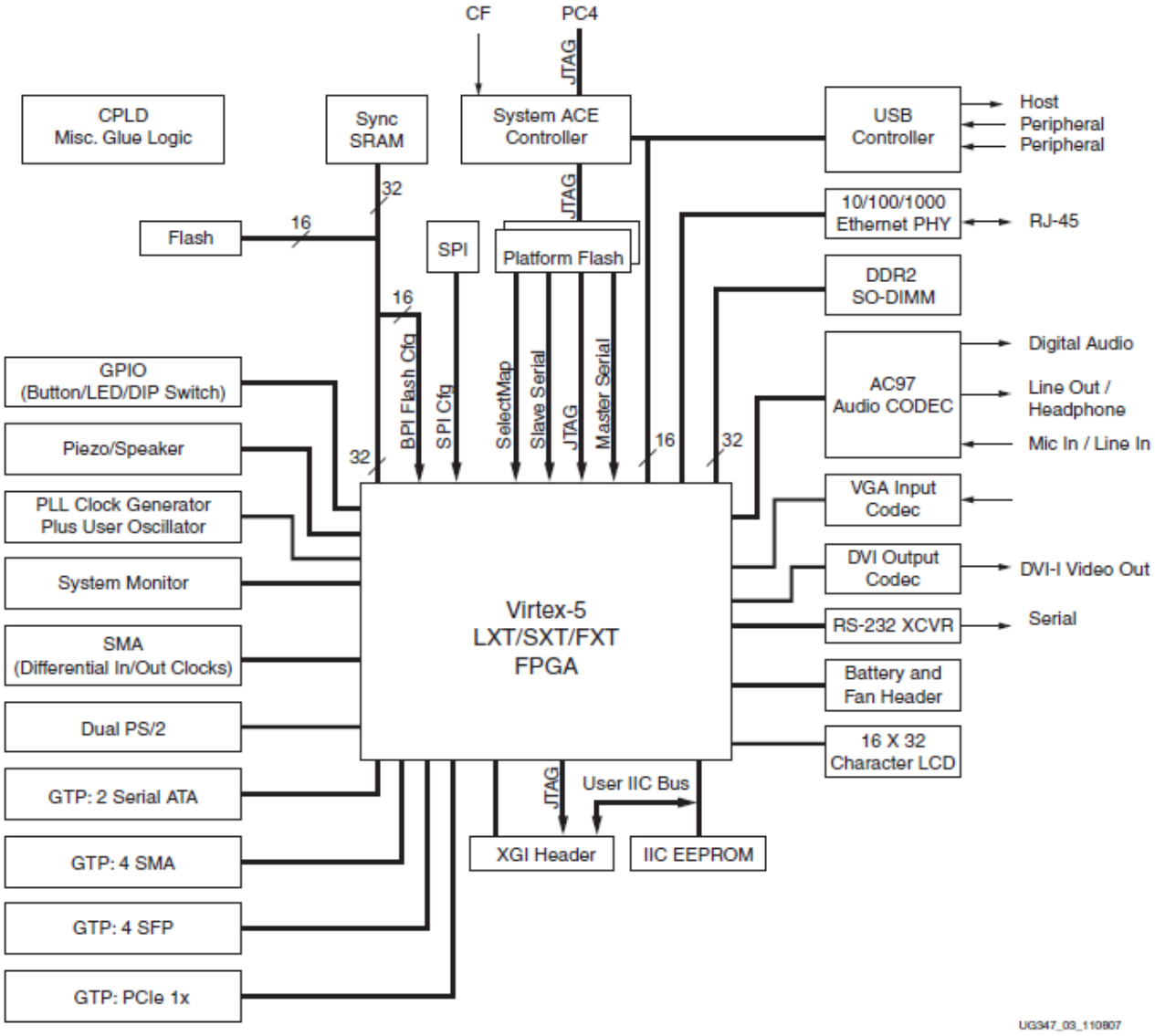

Figure 8 – Block Diagram of Xilinx ML507 Evaluation Platform

3.1.6 Interfacing evaluation board

The **DAQ Adapter** acts principally as an “add-on” module that is able to:

- perform an analog to digital conversion of an incoming A-FSK audio signal;
- feeds the 8-bit digitized data for the ML507 Xilinx board through the general purpose interface available on the board;

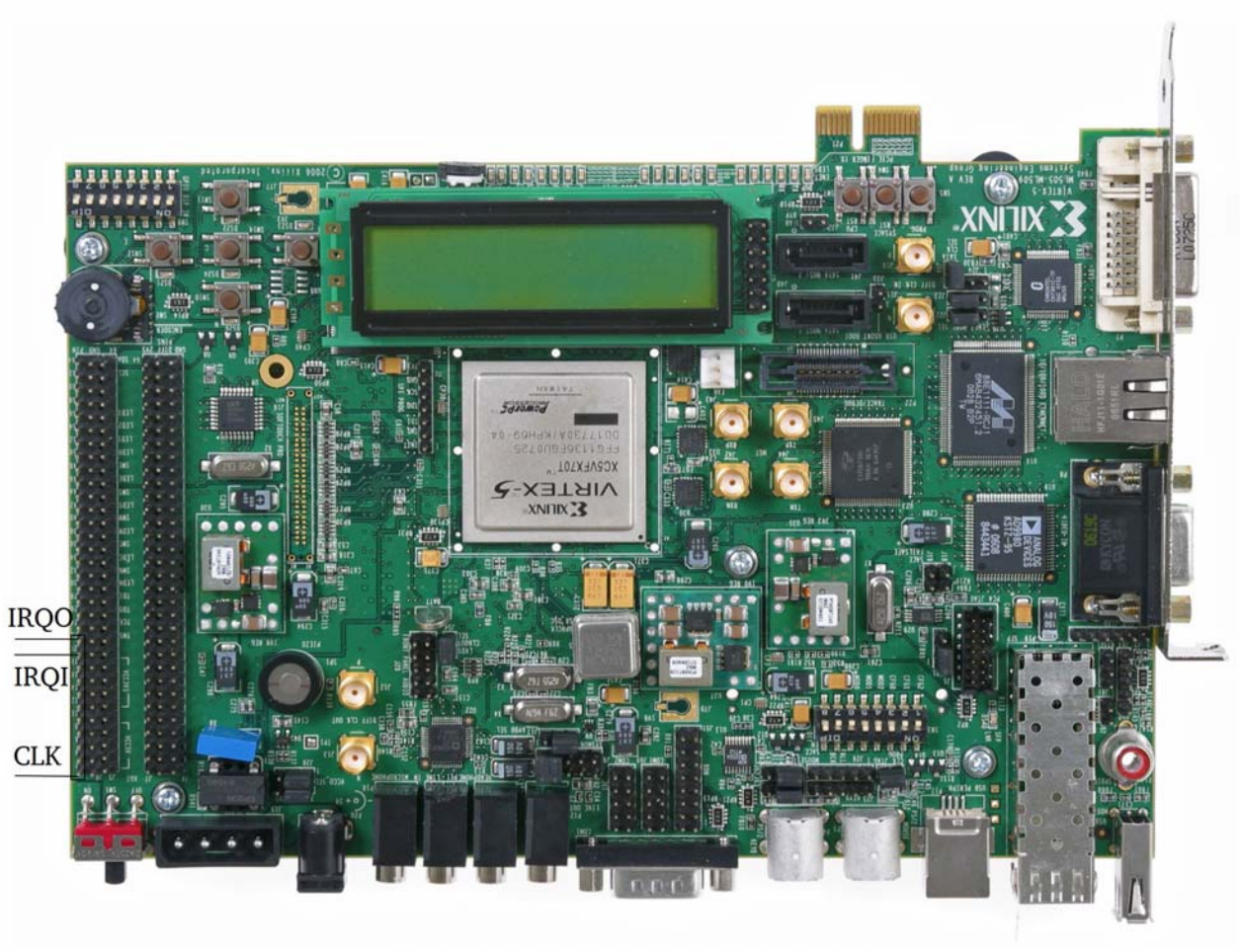

Figure 9 – The evaluation board

The next table shows the pin connection between the DAQ Adapter and ML507 Xilinx board general purpose interface:

Table 2 – Evaluation board pin connection list

Xilinx ML507	Net Name
J6.2	CLK
J6.4	D0
J6.6	D1
J6.8	D2
J6.10	D3
J6.12	D4
J6.14	D5
J6.16	D6
J6.18	D7
J6.20	IRQ IN
J6.22	IRQ OUT

3.2 Rugged High Performance Computing Node [ETH]

3.2.1 Power node HW/SW

Power Node will be a rugged embedded system, optimally designed in terms of dimensions, weight, power consumption and capable to work in harsh environmental conditions. The reference application context is defence/aerospace, ground mobile and airborne environments, addressing manned and unmanned applications where reliable high performance computing is required.

The Power Node will be based on a powerful computing architecture: a dual Intel Xeon 5500/5680 series (Quad core CPU) motherboard, with at least 6GB of on-board soldered DDR3 memory and a high data retention 80GB SSD drive. A high speed, high density FPGA device will also be present, providing easy adaptability and implementation of dedicated functions and special algorithms. It will offer a maximum processing power of 80GFlops.

In the following images the concept of the Power Node is described. The first image illustrates the form factor of the Power Node board and the positioning of the components on the board itself. The second image represents the board covered by a cold-plate that can be air or liquid cooled. The shape of this cold-plate is intended, at this stage of the project, only for descriptive purposes. The final version could be different.

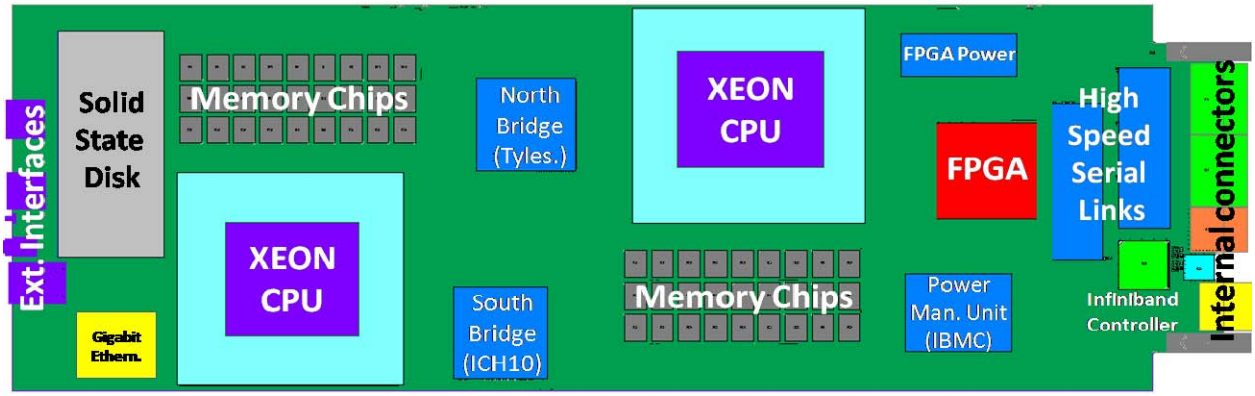

Figure 10 – Power Node board concept, without cooling heat sinks

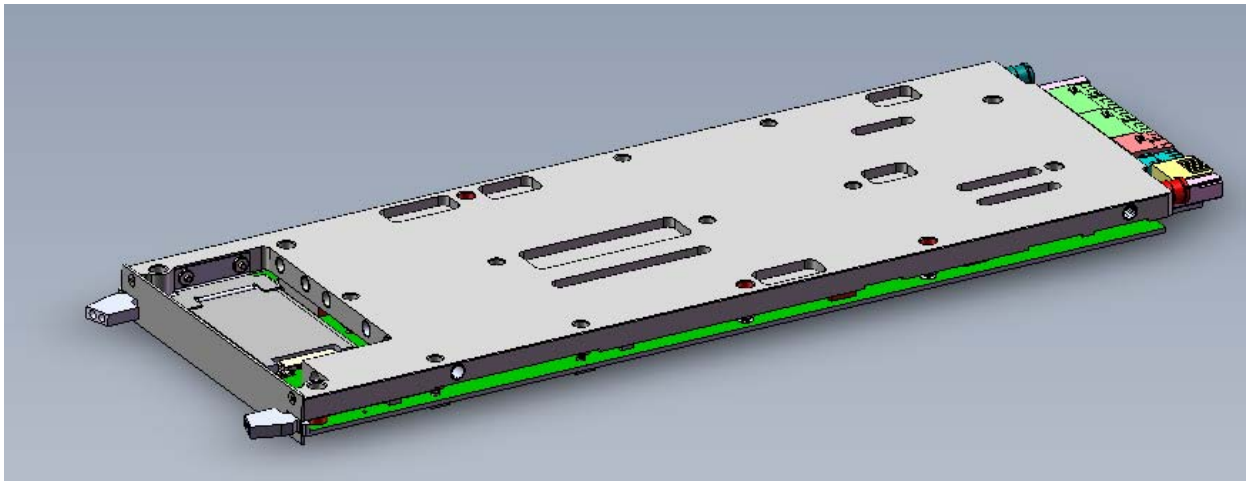

Figure 11 – Power Node board concept, with cooling heat sinks

1.1.1.1 Power Node software (OS, Protocol stack, Interfaces)

The software development for the Power Node will be mainly devoted to the adaptation of a commonly available Linux Distribution, in order to benefit from the richness of the features of a widely adopted operating system.

Regarding the OS the first choice will be “RedHat Enterprise Linux OS Verison 5.5 x86_64” which needs a license but is very well supported. Alternatively, if an open-source Linux distribution is required, the Power Node can support Linux distribution derived from RedHat, which are available for free but don’t have usually an excellent support. In this case, the operating system could be one of the following:

- CentOS
- Scientific Linux

In addition to the OS the porting of device drivers for the Infiniband networking interface and for the IBMC Board Management Controller will be provided.

The design of the FPGA firmware and software is intended to be implemented by the user of Power Node using ALTERA development tools:

- QUARTUS II (http://www.altera.com/products/software/quartus-ii/subscription-edition/qts-se-index.html)
- USB-JTAG programming/debugging tool (http://www.buyaltera.com/scripts/partsearch.dll?Detail&name=544-1775-ND)

As a starting point, many reference design, optimized for the same FPGA used in the Power Node, can be downloaded from Altera website. They reduce time to implement complex interface such as PCIe by means of pre-compiled building block.

To develop end-user applications, the final software development kit will contain the following additional tools:

- Infiniband OFED driver Stack supplied by Mellanox (basically standard OFED stack 1.5.1 pre-compiled). The package contains drivers and libraries for the InfiniBand interface and for the 10Gb Ethernet interface (http://www.mellanox.com/content/pages.php?pg=products_dyn&product_family=26&menu_section=34#tab-three)
- IPMI tools
- Scientific Computation Libraries from EPEL Repository (they need separate free licensing)
- Intel C/C++ and Fortran Compilers
- Intel Math Kernel Libraries (All the Mathematic primitives: FFT, Matrix calculations etc)
- Intel Integrated Performance Primitives (these are basically computational accelerators)
- Other Intel Libraries (these are proprietary libraries for example: treading building block)

1.1.1.2 Power Node hardware (Radio, Power, CPU, Interfaces, Sensing, extras (FPGA etc.))

The Power Node is a High Performance Platform based on Nehalem/Wesmare Xeon Intel dual-processor board with Tylesburg chipset; it is equipped with a high density FPGA and a high speed Infiniband controller, moreover there is an Ethernet Gigabit interface. Every component is supervised by a Power Management Controller Unit (IBMC).

The Power Node core architecture will consist of two Intel Xeon X5680 or X5570 CPUs, connected via Quick Path Interconnect (QPI), a dedicated low latency and high bandwidth bus capable of up to 6.4GT/s. Three channels of DDR3 memory are connected to each CPU, which integrates a high performance memory controller. The system hub (I/O Bridge) will be an Intel 5520 (Tylersburg) chipset and provides connectivity between the CPUs and the rest of the system; each CPU is connected to its Tylersburg with a QPI link. A Mellanox QDR ConnectX2 adapter is connected to the Tylersburg via one x8 PCIe 2.0 link: it provides a high Infiniband compliant connection. The hardware programmable part of the Power Node is represented by an Altera Stratix IV FPGA, which is connected to the Tylersburg with 2 x8 PCIe 2.0 links. Finally, the peripheral hub (Intel ICH10) is connected to the Tylersburg and provides the following additional peripherals:

- one optional SATA SSD, used to provide local fast and permanent storage
- one Zoar Gigabit Ethernet adapter
- 2x external accessible USB ports
- one Output Video Port
- one UART for low level debug

The independent, embedded controller for the Power Management (IBMC) allows the monitoring of each performance parameters, such as temperatures, voltages, etc. Access to these parameters can be done by the Power Node applications, locally and remotely over the network. The IBMC provides an SNMP interface to the Power Node and allows setting traps for specific events. It can also trigger and monitor the Power-On-Self-Test. In terms of remote control, the embedded IBMC permits the remote configuration of the Power Node through the network and additional remote configurability can be done through the FPGA.

The overall architecture of the Power Node is represented in the next figure.

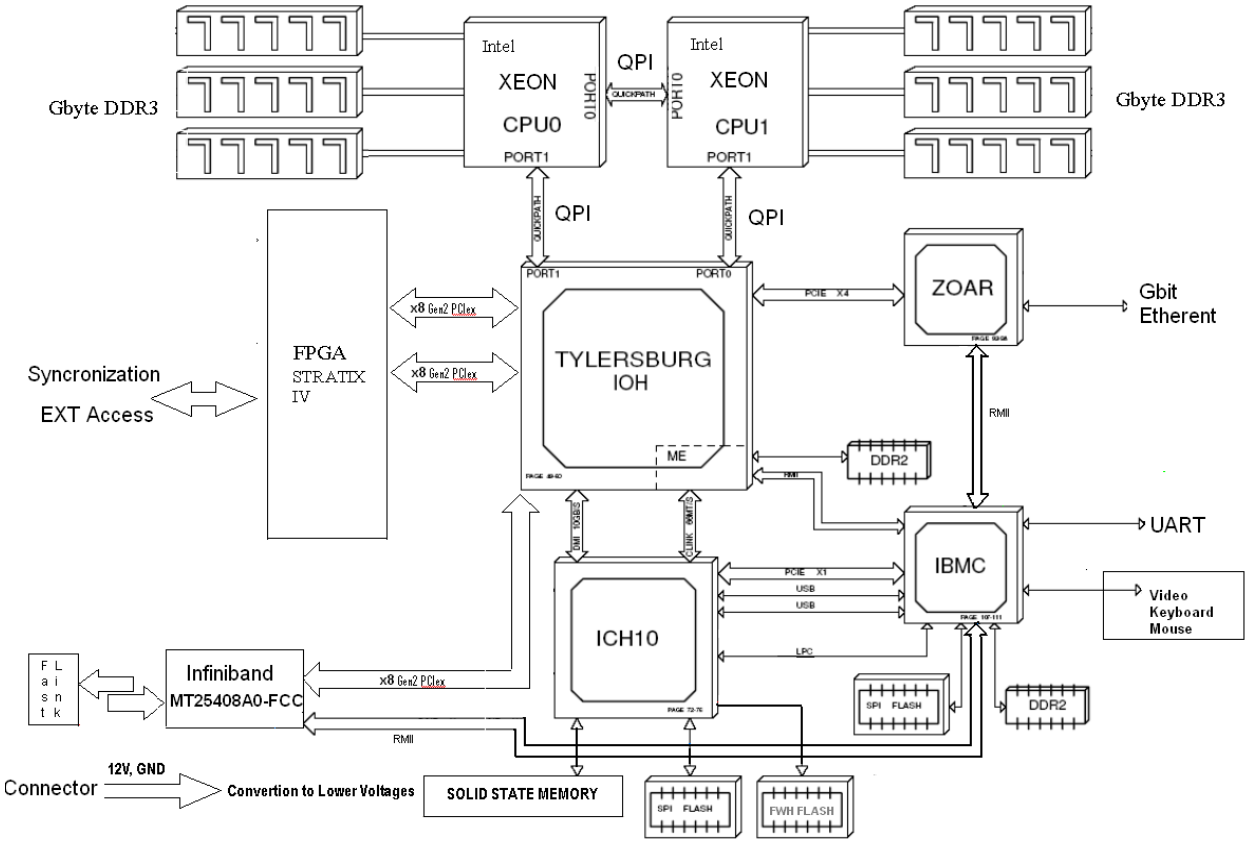

Figure 12 – Power Node architecture: high level description

The FPGA Processor is responsible for some security aspects. It includes a core logic that monitors the security of the Power Node. Tampering with the node triggers a protection mechanism in the security node that:

- physically disconnects any I/O and network
- deletes any data resident on the node
- initiates the physical destruction of the device itself by driving the power supply
- provides security features such as cryptographic capabilities through a dedicated core embedded in the FPGA
- more in general, the hardware supports the Intel AES-NI technology

The Power Node architecture has been conceived thinking also “composability”, in order to provide the possibility to build network of Power Nodes depending on the specific requirements of the specific application context. The Infiniband interface allows creating virtual 3D torus networks of Power Nodes, which are very efficient in terms of bandwidth and latency, and are capable of scaling up with no performance penalty. The torus network is managed by a network

processor implemented in the FPGA of each Power Node, which interfaces to the system hub through two x8 PCI Express Gen 2 connections, for a total internal bandwidth of 80Gbs. Thanks to the FPGA implementation, the torus network processor permits standard, ad-hoc and application-dependent collective communications. Finally, the I-O and network interfaces are programmable, in order to permit interfacing the system to multiple network and bus technologies and protocols, increasing in this way the potential scalability of the network.

The possibility to aggregate multiple identical units has an impact also on dependability, providing redundancy. The execution segregation through hardware virtualization allows for protection, monitoring, disabling and replacement of malfunctioning or compromised nodes. Moreover, in case of a fault, redundant hardware provides dependable operations. This is accomplished at the hardware level through duplication of the resource and at a functional level through aggregation of resources (spare Power Nodes).

1.1.1.3 Power Node Reconfigurability

The capability of the Power Node to reconfigure itself, at runtime, is offered by the use of “in-system programmable” devices such as an FPGA. This means that according to an environmental request, not only the software libraries can be dynamically loaded, but also the hardware accelerator configuration can be modified at any time. With configuration we intend the hardware logic previously programmed in the FPGA.

As shown in the following image, hardware silicon internal part of the FPGA are based on SRAM Logic Elements which consist of combinational logic attached to memory elements and they can be combined to implement any type of hardware function.

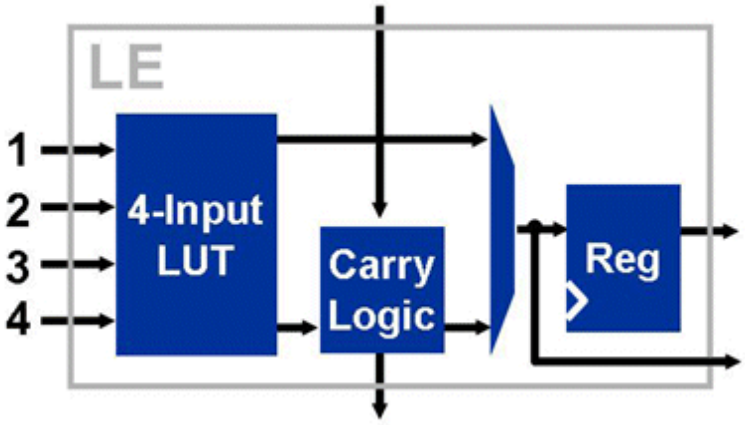

Figure 13 – internal structure of an FPGA logic element

Complex hardware functionalities can be designed with high level hardware description languages such as VHDL o Verilog or through schematic entry tools provided by development IDE.

Once the design has been completed and synthesized the development tool provide a binary file which can be written to the target device (FPGA) to update the configuration to the newer one.

The standard interface to access configuration registers of the FPGA is the JTAG port and it is used to write on it the binary file produced by the compiler.

The Power Node uses a USB-JTAG converter to grant OS the access to HW reconfiguration. The converter is integrated on the Power Node board. This solution has been adopted on both release of the prototype to simplify and improve the development and debug process. A second solution, that doesn't require the USB-JTAG converter, could be adopted in future versions of the prototype that will be closer to a final product. The current hardware already allows the implementation of this solution that, in terms of functionalities, is perfectly equivalent to the one adopted. This second solution is based on the direct reconfiguration of the FPGA through the PCi Express bus. A software application is capable to store the FPGA binary images into the Flash memory connected to the FPGA, and chooses the most suitable image depending on the threat identified. In this case, a specific operating system driver must be implemented to control the PCi Express bus and an engine, that acts as a bridge between the bus itself and the flash memory, must be implemented into the FPGA and added to the FPGA application specific logic.

The reconfigurability features offered by the Power Node can be used in a real application scenario as follows:

1. A threat is identified by proper application logic.
2. The application, depending on the threat, decides if a reconfiguration of the FPGA is required
3. The operating system stops processes that use the current hardware configuration
4. The application chooses the new configuration capable to face the threat
5. The selected configuration is written via JTAG to the FPGA

The operating system starts new processes associated with the new HW configuration.

1.1.1.4 Technical Specifications

In terms of technical specifications, the Power Node will feature:

- 2 Intel Xeon 5570/5680 CPUs at 2.93/3.33GHz
- At least 6GB RAM 1333MHz DDR3
- Optional FPGA device, which allows implementation of:
  - ✓ Hardware accelerator features (on board co-processing)
  - ✓ Synchronization network for multi node mode
- Custom processing units
- Optional 80GB 1.8" SATA SSD
- Independent sensor network and monitoring system
- Connectivity via two additional debugging board that will bring the signal on standard connectors for an easier access by the user
- QDR Infiniband port
- LAN 10/100/1000 Interface
- VGA Analog Video output
- 2x USB 2.0 host interface

Physical Specifications:

- Physical dimensions: 166mm h x 25,4mm w x 500mm d
- Weight: 2.2 Kg (with cooling system)

Power Specifications:

- Power consumption: 350W typical (420W Max)
- Power Supply Voltage 12V

4 Hardware and Software crypto technologies [CS, ATHENA, AS, THYIA]

The constrained resources of some of the system's components put serious limitations on the range of the available cryptographic primitives that can be used to secure it. Choosing the right cryptographic technology or technologies to address security will be made by a combination of research and testing different approaches and technologies in order to choose the best for the specific problem, taking into account power consumption, performance and security assurance regarding specific cryptographic tasks, such as signing, verification, encryption and decryption. Once the research and test phases are over, the chosen approach will be prototyped, thus allowing integration into the respective demonstration environment.

4.1 Cryptographic algorithms selection [CS]

Cryptography algorithms made a fundamental contribution for the provision of different systems security, inside the pSHIELD enabling technologies. In the next figure, the main enabling technologies are listed as well as their main relation with the above key concepts.

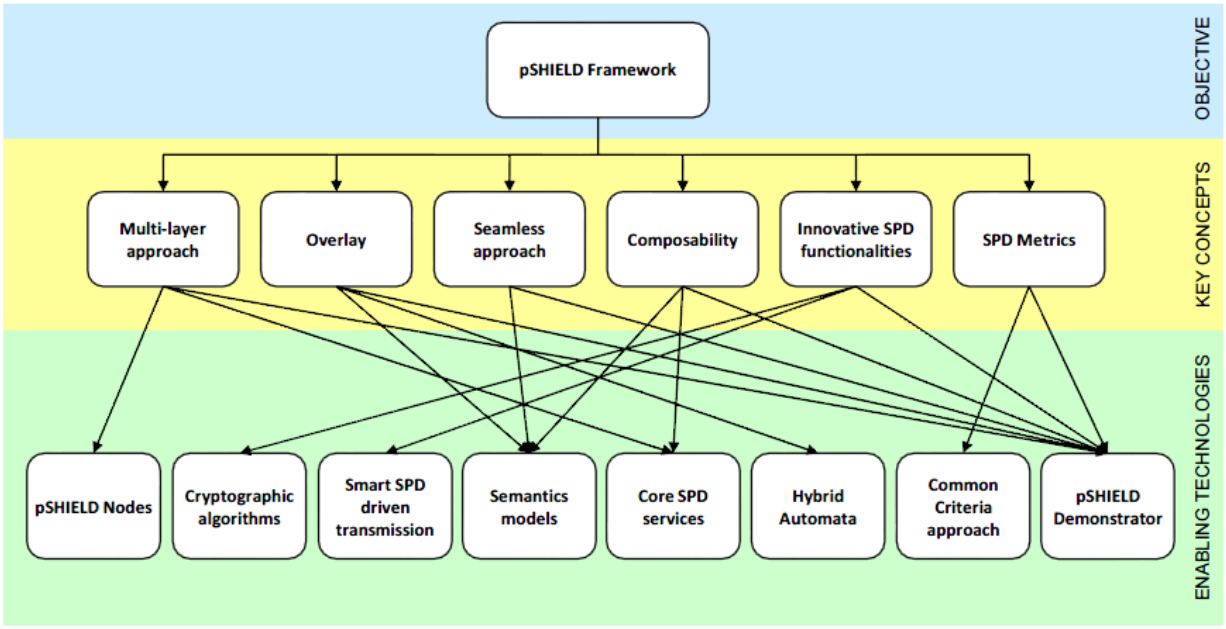

Figure 14 – pSHIELD enabling technologies in the framework

The Cryptography algorithms are under the innovative SPD functionalities, with responsibility of achieving the four goals: confidentiality, authenticity, data integrity and non-repudiation.

Understanding that the pSHIELD project is highly concerned with lightweight devices, we discuss the suitability of the studied algorithms accordingly.

The security provided through cryptographic means comprises mathematical cyphering algorithms and key management techniques. Common cyphering algorithms are divided into two types; asymmetric and symmetric. Key management, however, is influenced by different factors such as system's architecture and class of devices.

Having security as a main element in the sought SPD architecture, the study of available approaches to cryptography, presented in section 6.2.1 of SPD self-x and cryptographic technologies (Deliverable D3.4) becomes fundamental. This section highlights the main points of

that document, namely the description of different asymmetric and symmetric cyphering algorithms.

4.1.1 Asymmetric Cryptography

Asymmetric cryptography, also known as public key cryptography ([1], [2]), is based on the disposition of two types of keys, a public key and a private key, that are used in the cryptographic operations. Intuitively, the public key is made available by a given entity to potential senders while the private key is kept hidden by that entity. A message sent to an X receiver should be encrypted by X's public key where X can later decrypt it using its private key.

There are mainly three well-known types of asymmetric cryptography algorithms [3]; Elliptic Curve Cryptography (ECC), Rivest Shamir Adleman (RSA) and EL-Gamal. Depending on the target application and scenario specifications, implementations of the aforementioned approaches can be in software, hardware or a co-design of both.

1.1.1.5 Rivest Shamir Adleman

RSA [2] is the classical public key cryptographic algorithm (based on factoring) that has been popular for the last couple of decades. The essence of RSA is the intractability of solving the RSA problem.

The main operation in RSA is the modular exponentiation. That operation is normally broken down into a set of simpler operations such as modular multiplication, modular addition and addition operations. Several hardware designs for those operations have been proposed [4] as well as software implementations [5]. For completeness, Pretty Good Privacy (PGP) is an early adopter of RSA and has provided a packaged solution for email encryption and authentication. PGP uses the concept of Web of Trust for authentication through which signing and certification is done in a distributed manner. This is as opposed to the Public Key Infrastructure (PKI) [6] scheme where a certificate authority is used in a centralised manner to securely affiliate users to their public keys. PGP follows the standardised specification known as OpenPGP [7] where an open source implementation is provided in GnuPG [8].

1.1.1.6 Elliptic Curve Cryptography

ECC [9] makes use of the characteristics of the elliptic curve that can be defined on the Cartesian coordinate system by the formula $y^2 = x^3 + ax + b$. For different values of a and b different elliptic curves are produced where points (x, y) satisfying the equation fall on the curve in addition to a point at infinity. The public key is a given point on the elliptic curve where the private key is a pseudo random number. Other domain parameters are agreed upon between the communicating parties such as a generator point selected on the curve besides the values of a and b. Normally, the public key is computed by multiplying the generator point by the private key. The main idea behind the security provided by ECC is the infeasibility of computing the discrete logarithm of an elliptic curve element given only the public base point. For example, given that P and Q are points on the elliptic curve where $kP = Q$, assuming that k is a sufficiently large number, it is infeasible to find k it being the discrete logarithm of Q to the base P. Hence, point multiplication (e.g., k multiplied by any point on the elliptic curve) makes the essence of ECC.

The main advantage of ECC is its relatively small key size as opposed to some other public key cryptographic approaches [10]. Moreover, ECC with a key size of 160 bits proved to provide the same level of security as its counterpart RSA with a key size of 1024 bits. As the main primitive operation used in ECC, point multiplication has a major role in determining how efficiently fast the algorithm is regardless of the implementation type. Point multiplication is normally broken down into point addition and point doubling operations. Consequently, the rapidness of ECC is dependent on the nature of the algorithm used for carrying out these operations and the manner through which they are implemented, i.e., software, hardware or a combination of both.

The implementation of ECC (and many other cryptographic algorithms) should naturally take into consideration the application type and the consequent trade-offs required. Restrictions in terms of processing power, memory, energy, communication medium and possibly available area (in the case of hardware implementation) all take part in the set of design decisions for the sought implementation.

With the increasing pervasiveness of constrained embedded devices and their growing networked nature, more interest is being shown in stand-alone hardware ECC implementations [3]. For example, ECC hardware-based cyphers are recommended for the security needed in authenticating smart cards as they require less information exchange [10]. However, the cost of a pure hardware ECC implementation could hamper its adoption in the networked embedded systems domain. Consequently, hardware assistance through the extension of instruction sets becomes more favorable. In certain situations, the hardware provided is not flexible enough to new changes. As a result, pure ECC software implementations are inevitable and are needed to be carefully optimized in order to deal with the challenges posed by the constrained devices. These devices could have as low as 8-bit microcontrollers, 128 Kbytes flash memory, 4 Kbytes of SRAM and a similar amount for EEPROM. Where the speed of point multiplication is decisive in any ECC implementation, the software should avoid expensive operations such as inversion. For example, the use of Jacobian projective coordinates for point multiplication could be beneficial in this case.

1.1.1.7 EL-Gamal

In essence, cryptography relies on the infeasibility of certain mathematical problems. Where the cryptographic strength of RSA is based on the problem of factoring large numbers, EL-Gamal [11], named after its inventor Taher El-Gamal, is based on the difficulty of the discrete logarithm problem over the cyclic group G. EL-Gamal is essentially based on the Diffie-Hellman key agreement [12].

EL-Gamal encryption simply works as follows: Find a large prime number p that has a hard discrete logarithm problem and a generator g on Z^*p. An entity E1 then chooses a random exponent x that falls in $(0 < x < p - 1)$ as its private key and computes $X = g^x$ as the public key. Another entity E2 can encrypt plaintext message $m \in Z^*p$ and send it to E1 by choosing a random exponent y within $(0 < y < p - 1)$ and computing $Y = g^y$, $K = X^y$ to produce the cyphered message $C = K \cdot m$. E2 should then send to E1 the pair (Y,C). E1 can simply decrypt the received message by calculating $K = Y^x$ and $m = C/K$. Similarly to RSA, padding is needed in EL-Gamal in order to avoid several well-known attacks.

El-Gamal has been implemented in hardware such as smart cards as well as in software. However, hardware implementations need to be carefully designed to be tamper-proof especially in the case of protecting the pseudo random generator needed in EL-Gamal [13].

1.1.1.8 Asymmetric algorithm Discussion

RSA, ECC and EL-Gamal are the three most representative algorithms for well-known asymmetric cryptography approaches. In light of the pSHIELD project, lightweight devices are considered to be integral to the overall architecture. Hence, the suitability of the presented approaches is studied accordingly.

In essence, a number of official bodies have identified ECC as a suitable cryptographic approach to deal with lightweight devices. Standardizing bodies such as the International Standardization Group (ISO) [14], the American National Standards Institute (ANSI) [15], the Standards for Efficient Cryptography Group (SECG) [16] and the National Institute of Standards and Technology (NIST) [17], have all adopted ECC for lightweight devices [3]. In an energy analysis study comparing ECC and RSA implementations for a similar but slower hardware [10], ECC was showed to be a promising approach. The nodes used where the mica2dot with Atmel 128L 8bits CPU with the cc1000 Radio. The ECC with 160 bits key size (ECC-160), takes up to 1.61 seconds for point multiplication and 282 bytes of memory, while RSA with 1024 bits key

size (RSA-1024) takes up to 22 seconds for the modular exponentiation and 930 bytes of memory.

The relatively smaller key size required by ECC in order to provide a similar security level as RSA for example, is considered a major advantage in lightweight devices. This is due to the fact that ECC will hence save in memory, bandwidth and computational needs. To clarify, in a lightweight handshake scenario, RSA-1024 consumes 397.7 mJ in energy on the client side while ECC-160 consumes only 93.7 mJ [17]. This is a marked 76.5% less energy consumption by ECC-160 as opposed to RSA-1024 in handshake only. This is also similar to the energy consumption rates on the server side. Concerning the energy cost for digital signature, RSA-1024 consumes 304 mJ for signing, 22.9 mJ for verification as opposed to 22.82 mJ and 45.09mJ respectively for ECC-160. The latter's verification is based on 2 point multiplications where this can be reduced to roughly 1.2 point multiplications through advanced optimization which results in less energy consumption. It is worthwhile mentioning that, on a similar platform, digital signing in ECC is considerably faster than in RSA, (i.e., roughly 97% faster) and that in terms of verification, RSA is however faster, (i.e., 22% faster) [31]. Also, key generation is considerably faster in ECC as opposed to RSA. Moreover, there is a difference in performance between ECC and RSA in both encryption and decryption operations. In encryption, on a similar platform, RSA is roughly 70% faster than ECC while in decryption ECC is roughly 87% faster than RSA [31].

Comparing EL-Gamal to RSA, it transpires that EL-Gamal consumes considerably more energy during the encryption process [32]. Using a MIPS R4000 processor at 80MHz frequency, EL-Gamal consumes 134 mJ as opposed to its RSA counterpart that consumes only 0.81 mJ for encryption. However, decryption using El-Gamal using the same processor consumes only 0.94 mJ while RSA consumes 16.7 mJ. The results are based on the multiplications (128 bit) required for each algorithm. A similar result was observed under a different processor (StrongARM - 133 MHz) for both RSA and El-Gamal in terms of energy consumption. El-Gamal in this case consumes 123 mJ and 9.1 mJ for encryption and decryption respectively as opposed to 0.74 mJ and 15 mJ in the case of RSA.

It transpires that from among the most popular asymmetric cryptography algorithms, ECC emerges as a promising candidate for lightweight devices whether implemented in hardware [17] or software [16]. This is mainly due the small key size used while being able to maintain a high level of security.

4.1.2 Symmetric Key Cryptography

Symmetric ciphers use the same key or a pair of trivially-related keys (e.g., one is a linear transformation of the other) for both encryption and decryption of messages. Historically, symmetric ciphers precede their asymmetric counterparts and, although less versatile in their applications, they continue to be widely used due to the fact that they are typically several orders of magnitude faster, as well as, they can be implemented more efficiently. The main downside of symmetric key cryptography is the need to establish a secure communication channel for key exchange between the communicating parties before the actual communications can begin. As a result, asymmetric (public key) cryptography is often used to exchange symmetric session keys between the two parties and then to use a symmetric cipher to encrypt all subsequent communications.

Symmetric ciphers can be grouped into two broad categories: stream ciphers and block ciphers. The former combine a pseudo-random bit sequence with the plaintext (typically a XOR) and, thus, operate on individual bits or bytes of the plaintext, while the latter use fixed-size blocks of plaintext. Stream ciphers are typically faster and simpler to implement than block ciphers, both in software and in hardware, and are better suited for encryption of transmissions of streams of large amounts of data (e.g., video streams). However, stream ciphers have been reported to have serious security vulnerabilities when not used carefully. In particular, keys should never be reused otherwise the plaintext can be easily recovered.

Block ciphers use fixed-size blocks of plaintext, typically of 128 bits, and transform them in a sequence of operations, called rounds. Encryption of messages longer than the block size is done using a *mode of operation*, i.e., a technique of partitioning the plaintext into a sequence of blocks and then chaining their encryption to construct the cipher text of the entire message. Encryption of plaintexts smaller than the block size is done using a padding scheme.

Symmetric ciphers have to be used in embedded system applications due to performance considerations since the use of public key cryptography puts significant strain on both computational and energy resources. Not all symmetric ciphers, however, have comparable performance characteristics. The ciphers may differ in their security strength, in the achievable data throughput, in ROM/RAM requirements of their software implementations or in the number of gates required to implement them in hardware. Therefore, a question arises as to the choice of symmetric cipher for use in the pSHIELD middleware.

Evaluation of cryptographic primitives is a process that typically takes several years of analysis by expert cryptologists in which security properties, as well as performance characteristics are analysed and compared. There have been several calls for ciphers organized in the last decade and the most notable include the standardization effort for the Advanced Encryption Standard (AES), the European NESSIE and eSTREAM projects, and the Japanese government's CRYPTREC project. The following sections overview the selected finalists of these competitions with the goal of using the recommendations for the pSHIELD framework and demonstrators.

In the following sections, the pSHIELD recommended symmetric algorithms are presented and we start the presentation with Rijndael AES.

1.1.1.9 Rijndael AES

Designed by two Belgian cryptographers, Joan Daemen, and Vincent Rijmen, Rijndael is the winner of the AES competition. This competition, organized in 1997 by , the National Institute of Standards and Technology of the United States (NIST) had the goal of find a replacement for the Data Encryption Standard (DES), the dominant symmetric cipher of the time. DES is a block cipher with 56 bit-long keys that had been in official and widespread use since 1976. In the nineties, the cipher was becoming increasingly vulnerable to brute force attacks due to its somehow limited key space.

Since AES competition, the name Rijndael is used to denote the original and more general cipher definition while the name AES is used to refer to its standardization of the cipher, i.e., a version that permits only a small range of possible key and block lengths. In the rest of the discussion, we will interchange both names whenever it is clear from context.

AES uses a fixed block size of 128 bits and it supports three key lengths of 128 bits, 192 bits and 256 bits. Encryption is done in 10, 12 and 14 rounds for the different key lengths respectively. The cipher follows the design principle known as substitution-permutation network, i.e. it is defined as a sequence of alternating mathematical transformations called substitution boxes (S-boxes) and permutation boxes (P-boxes), which are applied iteratively to portions of the plaintext block and to derivations of the encryption key (so called, round keys).

The cipher enjoys a very neat mathematical formulation and, although it is considered very secure, some cryptologists believe that this property could be the cause of some successful attacks in the future. Until 2009 the only realistic attacks proposed were side channel attacks, hence they did not invalidate the cipher's design [34]. Later, a related-key attack was proposed that requires a pair of related keys and 2^{39}-time to break the nine-round version of AES 256, and a related sub-key attack on the ten-round version of AES 256 that needs 2^{45}-time [35]. These attacks do not pose a serious security risk since they require access to the same cipher text encrypted under two related keys, as well as because they are not applicable to the full-round AES. Nevertheless, they decrease the available security margin of the cipher, indicating that the standard might be revised in the coming years by, for example, extending the number of encryption rounds. Note also that no realistic attacks on AES 128 have been proposed to date.

Following extensive evaluation by the AES committee, the cipher was reported to have the best performance characteristics, combining very good security parameters with high encryption rates and low RAM/ROM overhead on both high-end processors and resource-limited smartcards [36].

1.1.1.10 CAMELLIA

Camellia is a 128-bit block cipher that was designed by Mitsubishi and NTT and approved by both the NESSIE as well as the CRYPTREC projects. Also, the cipher was standardised by ISO/IEC and the Internet Engineering Task Force (IETF) approved the cipher for use in OpenPGP and in the SSL/TLS, S/MIME and IPsec protocols. The cipher supports the same key lengths, block size and it follows the same general design scheme as AES. Camellia is, therefore, widely regarded as being comparable to AES both in terms of security and performance. The cipher is patented but royalty-free licence is granted for all uses.

Due to the similarity in design to AES, much of the security analysis done for AES applies also to Camellia. However, some cryptologists hold the view that the neat algebraic representation of both ciphers might be the source of their potential insecurity. Despite this critique, the NESSIE project panel concluded that no serious flaws could be found in the design of the cipher [42].

Camellia has very low memory consumption and the cipher can be used on such memory-constrained devices as smartcards [43]. However, although its designers claim that Camellia's encryption performance comparable to that of AES, comparisons done for the NESSIE project show that, although fast in general, the cipher can be even two times slower than AES on certain CPUs. On the other hand, Camellia's key setup procedure is uniformly faster than that of AES [31].

1.1.1.11 Salsa20/12

Salsa is the best-ranked cipher in the eSTREAM portfolio. The cipher uses 256-bit keys and 128-bit initialization vectors (of which 64 bits constitute the random nonce and the remaining 64 bits denote the position in the stream, thus allowing for rewinding of the input stream). The original cipher formulation uses twenty encryption rounds and it is, thus, referred to as Salsa20. Lower-complexity variants have also been submitted to the project, namely Salsa20/12 and Salsa20/8 of which Salsa20/12 was selected by the project because it offered the best balance between security and efficiency [65].

Although the cipher's security received considerable attention due to the simplicity and scalability of its design, no realistic attacks could be found. However, a modification was later added to the Salsa's design that extended the nonce length from 64 bits to 192 bits in order to further improve its security. The new cipher is known under the name XSalsa20 and its authors claim that the modification provably improves its security [71].

1.1.1.12 Symmetric algorithms Discussion

Symmetric ciphers are an important tool of modern cryptography. The main reason for their continued use is the need for security in spite of constrained resources because symmetric ciphers are orders of magnitude more efficient than the asymmetric ones.

Security can be a computationally demanding task not only on low-end or low-power embedded systems. More powerful systems such as web servers or high-bandwidth routers are used in applications that process large amounts of traffic and the choice of encryption algorithms affects also their performance. From [2] it is clear that providing security can be demanding even for high-end systems and that the efficiency of encryption cannot easily be remedied in many applications through the use of more capable hardware.

Although the use of symmetric cryptography is necessary, public-key cryptography still has its applications in the embedded world. The popular approach is to get the best of both worlds, i.e., to use public-key cryptography in situations that benefit from the use of public and private keys such as, for example, authentication and to use symmetric cryptography for fast encryption of data streams. The approach carries an additional benefit of solving the problem of secure symmetric key distribution: when both parties trust each other's public keys, symmetric session keys can be created and securely exchanged by encrypting them with the public keys.

Selecting the best cipher to use is a very difficult task because, apart from encryption efficiency, the main criterion is always security. Assuring the security of a cipher, however, is a lengthy process that involves detailed analysis by expert cryptographers, as well as performance evaluation on numerous hardware platforms. The experiences of the AES contest and the NESSIE, CRYPTREC and eSTREAM projects show that several years might not be enough to reliably evaluate the security of a cipher because finding new attacks is akin to making scientific inventions and these have long been shown to be unpredictable.

Due to resource limitations, as well as due to the need to build on and to reuse the existing work, the pSHIELD project should use a symmetric cipher that has very good security record and that has been exposed to cryptanalysis by experts over a number of years. Thus, the best choice is to focus on publicly -reviewed ciphers such as those submitted to the AES contest and to the NESSIE, CRYPTREC and eSTREAM projects.

The next selection criterion after security is efficiency. The following metrics can be used to assess efficiency of a symmetric cipher:

- Encryption/decryption speed (in processor cycles).
- Key setup cost (in processor cycles).
- Memory efficiency (RAM/ROM requirements of smallest implementations).
- Hardware implementation size (in the number of gates required).

Each of these metrics on its own does not give the full information about cipher's performance because different usage scenarios have different requirements. For example, the performance of securing point-to-point streaming data mainly depends on encryption speed but the security overhead for irregular packetized traffic is likely to be dominated by the key setup cost. Also, there are fast ciphers whose implementation takes up more memory than small embedded devices can allocate, as well as not all ciphers that perform well in software offer the same relative advantage when implemented in hardware (or vice versa). Finally, there are ciphers whose performance results are roughly uniform throughout a variety of platforms and there are those that perform better on, for example, high-end CPUs due to the use of specialized processor instructions.

The pSHIELD project aims at heterogeneous, networked and hierarchical embedded systems and such systems typically comprise devices of radically different capabilities, including ones with severely limited resources. Communication patterns may vary depending on the network nodes in question. For example, low-power sensor nodes will likely use sporadic packetized transmissions while larger devices, such as CCTV cameras, will have more resources and will need to transmit relatively large quantities of data on a continuous basis.

These requirements are quite stringent and there are essentially two ways the problem of communication security can be solved: either several different specialized ciphers are used by different sectors of the system or a fast, secure and efficient cipher is found for universal use across the entire network. The former approach is not preferred due to the increased cost and difficulty of implementing different ciphers while the latter might be difficult to realise on its own.

Although direct comparison of the performance of all symmetric ciphers presented in this document is difficult due to the lack of studies that would reliably compare all of them, certain

patterns can be found when comparing results presented by the AES, NESSIE, CRYPTREC and eSTREAM committees. In particular, Rijndael seems to be the fastest cipher of those selected by the project committees (hence of those considered secure). Rijndael's performance is uniform across different platforms, the cipher has small key setup cost and it can be efficiently implemented in software on such resource-constrained platforms as smartcards. The following is a quote on the Rijndael's performance [36]:

> Rijndael's assembly language on both the Pentium and Pentium Pro processors is about 300 clocks per block. Unlike RC6 and Mars, there are no known CPU platforms (8-bit or 32-bit) on which Rijndael's relative performance would be unduly negatively affected or on which timing attacks would be possible.

Rijndael was selected by the AES committee as the best cipher of all submissions, which offered both very good security parameters, as well as excellent performance results. The cipher was also used by the NESSIE and CRYPTRECT committees as a point of reference for the performance comparisons of their submissions. The NESSIE project included Rijndael in their performance evaluation framework and, according to the presented results, the cipher was the fastest among all secure submissions, as well as it was one of the fastest among all of the submissions [31]. The only cipher that came close to Rijndael in terms of performance (in all of the metrics) was Camellia. The cipher might be considered Rijndael's main competitor due to the fact that it offers somehow stronger security level (it uses a larger number of rounds) and because of its on-going standardization effort. Also, Camellia has faster key setup procedure making it better suitable to the use in low-power networks.

As far as the security is concerned, the 128-bit version of Rijndael, known as AES 128, has a very good record although the quality of the new attacks is steadily improving. The attacks, however, are not likely to threaten the cipher's security in the next several years and, thus, it seems reasonable to use the cipher for the pSHIELD project. Also, the widespread use of the cipher results in better public exposure leading to either a stronger security record (if no practical attacks are found) or to quick discovery of flaws.

Although Rijndael was rated to be the fastest block cipher from among those considered safe, stream ciphers offer even greater encryption speeds. For example, the eSTREAM performance measurements rate Rijndael to be two to three times slower than Salsa20/12 on many hardware platforms [80]. Stream ciphers, however, have to be used with care if they are to be used to replace block ciphers. In particular, the same key must never be used twice. Also, stream ciphers are vulnerable to substitution attacks that allow the attacker to change the contents of a message without decrypting its contents if the attacker knows the structure of the message. This property makes stream ciphers unsuitable for sporadic packetized transmissions in which nodes repeatedly send small messages of fixed structure (e.g., detection reports in WSN). On the other hand, stream ciphers would typically be a better choice for such streamed data transmissions as, for example, video or sound feeds. Therefore, we consider it reasonable to use stream ciphers in the pSHIELD project, but only in such applications as CCTV data streams. Following the eSTREAM project recommendation, we recommend the Salsa20/12 stream cipher for use in the project as it offers the best balance between security and efficiency.

Finally, due to the on-going research in cryptology, it is impossible to choose the perfect cipher that would stay secure and efficient over decades. Hence, the pSHIELD architecture should be modular and it should permit installation of an alternative cipher if new cryptanalytic breakthroughs render it insecure.

4.2 Cryptographic key exchange [ATHENA]

In course of the work performed in WP3, specific activities will be addressed to implement and deploy a new cryptographic key exchange algorithm called “Controlled Randomness” that limits the need of frequent key exchange between the communicating parties, thus boosting the robustness of the underlying cryptographic operations against cryptanalysis, while keeping the performance cost in acceptable, and in some cases favourable, levels.

We focus on the problem of key replacement and exchange. Periodic changes of cryptographic keys is a necessary operation as to ensure (i) minimal exposure of plain data in case of key compromise and (ii) minimal collection of encrypted data under the same key, as to harden cryptanalytic attacks. Designers of secure systems rely on the fact that keys are periodically changed, in order to maintain the high level of security offered by the cryptographic primitives. Key replacement operations define the rekeying period for an algorithm (how often the key should be changed) and the behaviour of the system during transition periods where both an old and a new key may be valid. This can happen for example in a loosely synchronized environment, where some clients can receive out-of-order packets from the servers.

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 approach we propose has three desirable properties for protecting against cryptanalytic attacks: (i) the attacker must find which packets are encrypted under the same key, in order to then mount a specific cryptanalytic attack, (ii) consecutive packets that may carry redundant information are encrypted under a different key, and (iii) the disclosure of one key does not reveal the contents of a whole session; even if an attacker can know the time moments that the compromised key is used, he cannot resemble a whole session but only sketchy details of it.

These properties allow to extend a key's lifetime, both in terms of time of use and volume of data encrypted under the same key. This extension of lifetime is desirable in many environments, since it achieves a higher level of security utilizing the same cryptographic primitives, it requires less frequent control messages to be exchanged, and it requires fewer management operations.

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 an novel way to achieve increased security with minimal hassle

It has been already shown that CRP leads to efficient implementations in a variety of consumer embedded systems. The goal is to apply this knowledge to the pShield concept and refine the model through its practical implementation within the scope of the demonstrator.

4.3 Cryptographic libraries selection [CS]

In this section, we present the selected pSHIELD cryptographic and numeric open-source libraries.

The initial list of open-source public domain or similar license (GPL, AGPL, LGPL, Boost, BSD, WTF, dual, special or others) cryptographic libraries and the relevant associated numeric libraries evaluated is presented in the next figure. Our focus was nesC, C, C++ and Java implementations.

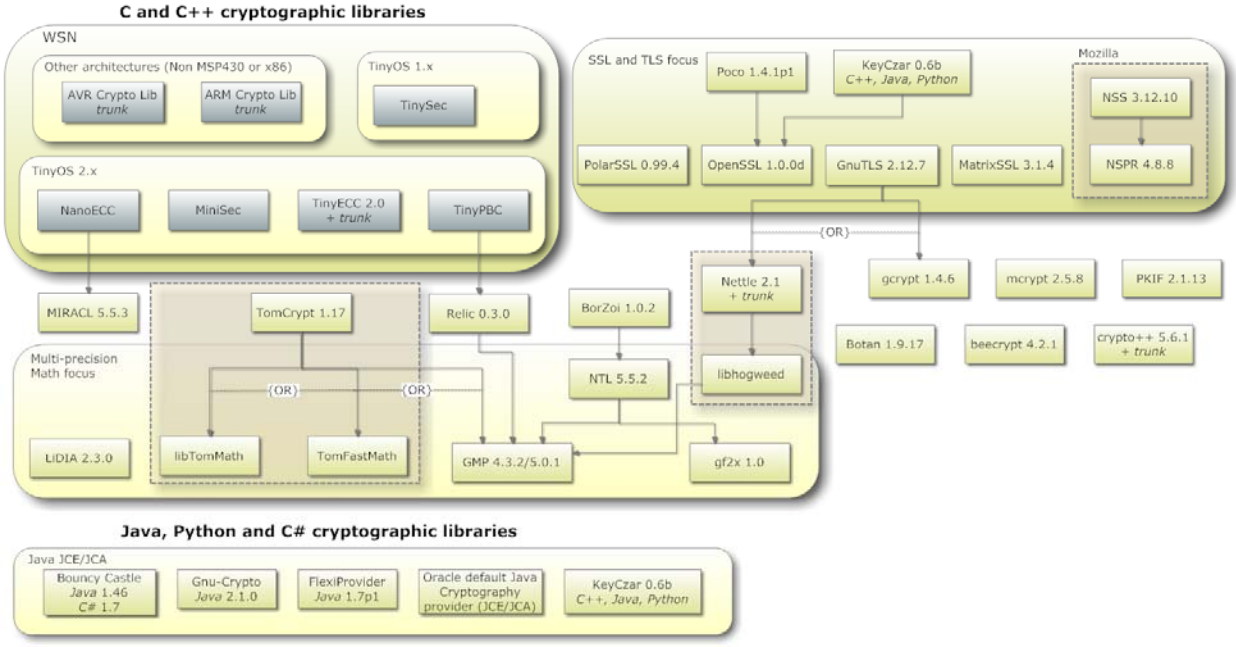

Figure 15 – Open-source cryptographic Libraries with dependencies

In the previous diagram, we presented the cryptographic libraries organized with nesC, C and C++ implementation grouped together and Java, Python and C# implementations on the bottom. The nesC implementations are only inside WSN group. Some libraries have focus on SSL and TLS and are used in browsers, ssh and httpd servers. One example is Network Security Services (NSS), used in several web browsers like Mozilla Firefox. NSS is one of the FIPS certified cryptographic module libraries for some binary versions.

Other libraries supporting public key cryptography depend on big number libraries like GMP. MIRACL and Relic-toolkit have multi-precision math functions but have also strong public key scheme support and for that reason were not included in the same group as GMP.

Some libraries like TomCrypt, Poco or Nettle are in fact a group of libraries, and may depend on additional external libraries to get better performance. Additional dependencies for external libraries like zlib were not added, in order to limit the compromised visibility and understanding of the diagram.

At the bottom, some Java cryptographic providers like Bouncy Castle, Gnu-Crypt and FlexiProvider are also presented. KeyCzar is the only duplicated box, since is a Java provider and also a C++ cross-platform library.

In the top left part, the wireless sensor network (WSN) specific libraries and extensions are presented. The TinyOS 1.x dependent TinySec where not evaluated since it's obsolete.

We present each selected cryptographic library in the next section. Other libraries from the previous figure are presented in detail in section 6.2.3.1.3 from SPD self-x and cryptographic technologies (Deliverable D3.4).

4.3.1 BeeCrypt

BeeCrypt is a cryptography library that contains highly optimized C and assembler implementations of many well-known algorithms including Blowfish, MD5, SHA-1, Diffie-Hellman, and EL-Gamal. Unlike some other crypto libraries, BeeCrypt is not designed to solve one specific problem, like file encryption, but to be a general purpose toolkit which can be used in a variety of applications. There are also no patent or royalty issues associated with BeeCrypt.

4.3.2 Botan

Botan is a C++ class library for performing a wide variety of cryptographic operations. Botan is released under the FreeBSD license.

4.3.3 Crypto++

Crypto++ (also known as CryptoPP, libcrypto++, and libcryptopp) is a free and open source C++ class library of cryptographic algorithms and schemes written by Wei Dai. Crypto++ has been widely used in academia, student projects, open source and non-commercial projects, as well as businesses. Released in 1995, the library fully supports 32-bit and 64-bit architectures for many major operating systems, including Apple (Mac OS X and iOS), BSD, Linux, Solaris, and Windows. The project also supports compilation under a variety of compilers and IDEs, including Borland Turbo C++, Borland C++ Builder, CodeWarrior Pro, GCC (including Apple's GCC), Intel C++ Compiler (ICC), Microsoft Visual C/C++, and Sun Studio.

Crypto++ ordinarily provides complete cryptographic implementations, and often includes less popular and frequently-used schemes. Examples of included implementations are Camellia (ISO/NESSIE/IETF approved block cipher) and Whirlpool (ISO/NESSIE/IETF approved hash function).

Additionally, the Crypto++ library sometimes makes proposed and bleeding edge algorithms and implementations available for study by the cryptographic community. For example, VMAC, a universal hash-based message authentication code, was added to the library during its submission to the Internet Engineering Task Force (CFRG Working Group); and Brainpool curves, proposed in March 2009 as an Internet Draft in RFC 5639, were added to Crypto++ 5.6.0 in the same month.

The library also makes available primitives for number theoretic operations such as a fast multi-precision integers; prime number generation and verification; elliptical curves and finite field arithmetic, including GF(p) and GF(2n); and polynomial operations.

Furthermore, the library retains a collection of insecure or obsolescent algorithms for backward compatibility and historical value like MD2, MD4, MD5, Panama Hash, DES, ARC4, SEAL 3.0, WAKE, WAKE-OFB, DESX (DES-XEX3), RC2, SAFER, 3-WAY, GOST, SHARK, CAST-128, and Square.

4.3.4 Nettle

Nettle actually consists of two libraries, libnettle and libhogweed. The libhogweed library contains those functions of Nettle that uses big number operations, and depends on the GMP library. With this division, linking works the same for both static and dynamic libraries.

If an application uses only the symmetric crypto algorithms of Nettle (i.e., block ciphers, hash functions, and the like), it's sufficient to link with -lnettle. If an application also uses public-key algorithms, the recommended linker flags are -lhogweed -lnettle -lgmp. If the involved libraries are installed as dynamic libraries, it may be sufficient to link with just -lhogweed, and the loader will resolve the dependencies automatically.

Nettle is distributed under the GNU General Public License (GPL).. However, most of the individual files are dual licensed under less restrictive licenses like the GNU Lesser General Public License (LGPL), or are in the public domain. This means that if you don't use the parts of nettle that are GPL-only, you have the option to use the Nettle library just as if it were licensed under the LGPL. To find the current status of particular files, you have to read the copyright notices at the top of the files.

4.3.5 LibTom

The LibTom Projects are open source libraries written in portable C under WTFPL. The libraries supports a variety of cryptographic and algebraic primitives designed to enable developers and students to pursue the field of cryptography much more efficiently. Currently the projects consist of three prominent libraries (LibTomCrypt, LibTomMath and TomsFastMath) which form the bulk of the source contributions.

Along with the source contributions, the LibTom projects also aim to serve an educational capacity. The libraries are very well commented, with clear and concise source.

All LibTom Projects are under WTFPL and free for all purposes.

4.3.6 Libgcrypt

This is a general purpose cryptographic library based on the code from GnuPG. It provides functions for all cryptographic building blocks: symmetric ciphers (AES, DES, Blowfish, CAST5, Twofish, Arcfour), hash algorithms (MD4, MD5, RIPE-MD160, SHA-1, TIGER-192), MACs (HMAC for all hash algorithms), public key algorithms (RSA, EL-Gamal, DSA), large integer functions, random numbers and a lot of supporting functions.

4.3.7 Poco

The Poco is a modern open source C++ class libraries and frameworks for building network- and internet-based applications that run on desktop, server and embedded systems. It has a Boost License.

4.3.8 PolarSSL

PolarSSL is a light-weight open source cryptographic and SSL/TLS library written in C. PolarSSL makes it easy for developers to include cryptographic and SSL/TLS capabilities in their (embedded) applications with as little hassle as possible. Loose coupling of the components inside the library means that it is easy to separate the parts that are needed, without needing to include the total library. This makes PolarSSL ideal for supporting SSL and TLS in embedded devices. PolarSSL is written with embedded systems in mind and has been ported to a large number of architectures, including ARM, PowerPC, MIPS and Motorola 68000.

4.3.9 TinyECC

TinyECC is a software package providing Elliptic Curve Cryptography based PKC operations that can be flexibly configured and integrated into sensor network applications. It provides a digital signature scheme (ECDSA), a key exchange protocol (ECDH), and a public key

encryption scheme (ECIES). TinyECC uses a number of optimization switches defined in the Makefile, which can turn specific optimizations on or off based on developer's needs.

TinyECC is intended for sensor platforms running TinyOS-2.x. The current version is implemented in nesC, with additional platform-specific optimizations in inline assembly for popular sensor platforms. It has been tested on MICA2/MICAz, TelosB/Tmote Sky, BSNV3, and Imote2. TinyECC 2.0 supports SECG recommended 128-bit, 160-bit and 192-bit elliptic curve domain parameters.

4.3.10 Conclusions

The main objective of this evaluation is the cryptographic libraries selection for the pSHIELD scenario. This evaluation was performed In the SPD self-x and cryptographic technologies (Deliverable D3.4) with security characteristics quantification in the Cryptographic Algorithm section (**Błąd! Nie można odnaleźć źródła odwołania.**) and with performance characteristics measurement during the previous section (**Błąd! Nie można odnaleźć źródła odwołania.**).

1.1.1.13 Low Power node conclusions

The AES Rijndael cipher is suitable for implementation on memory constrained devices and is supported in hardware by several wireless nodes including Telosb, as well as it offers high encryption and decryption rates for both continuous stream data traffic and sporadic communications. Finally, the cipher can be used to create reliable message authentication codes thanks to the ability of running it in the cipher block chaining mode (CBC-MAC).

During the evaluating of asymmetric and symmetric cryptographic implementations, we've detected several bugs, proposing some corrections to open-source libraries like PolarSSL and TinyECC that were accepted, resulting in changes on the last trunk version.

The WM-ECC, TinySec and MiniSec only supports tinyos 1.x. The TinyPBC supports the mica2 and micaz motes, with ATmega128 processor but not Telosb motes (with MSP430).

The AVR-Crypto-Lib and ARM-Crypto-Lib enable cross-compilation to different embedded systems than Telosb motes. The Telosb support for TinyECC could be improved with additional elliptic curve domain parameters. Additional libraries referred in academic research (like WM-RSA) are not open-source and/or the code was not public available.

Any additional pSHIELD supported wireless sensor network platform needs research and validation, since from our experience in Tinyos, the interoperability between systems cannot be taken for granted.

We advise that AES (Rijndael) and Camellia are two good candidate symmetric cipher for use in the pSHIELD scenario. Our preference goes to the first, since AES Rijndael is hardware supported on the Telosb radio chip (cc2420). Our asymmetric ciphers candidates for pSHIELD are 3 elliptic curve schemes: ECIES, ECDSA and ECDH all implemented in TinyECC. Every proposed ECC schemes have different application in Encryption/Decryption, Signing/Verification and Key exchange.

1.1.1.14 Power node conclusions

KeyCzar and borZoi were discarded due to halted development (borZoi), unsolved installation issues (KeyCzar) or poor support (all). The Supercop is a solid benchmark tool with rapid changes and frequent new version releases focus on the bare cipher implementation. Nevertheless, it took several days to evaluate the system, with all the compiler optimization flags coverage. Our choice was more agile, since we could see some fast results when we updated the trunk version of some libraries or when we changed the dependencies of some cryptographic libraries.

Libmcrypt, Beecrypt, Botan, Crypto++, Nettle and TomCrypt were the most generic C/C++ libraries with a good group of symmetric and asymmetric ciphers supported. Nevertheless, the mode of operation available in each library varies a lot. Beecrypt 4.2.1 supports ECB, CBC and CTR for AES operation mode but Botan 1.9.17 supports CBC, CBC-PKCS7, CBC-CTS, CBC.BE, EAX, OFB, CFB and XTS modes.

The performance generally rises as recent versions of the same library incorporate additional optimizations such the use of assembly code, the use of special purpose instructions available in some processors or in the motherboard or multi-thread use taking advantage of the multi-core processors. There were exceptions as Botan 1.8.11 process Gost at 33,02 and Idea at 19,67 and Botan 1.9.17 process Gost at 21,9 and Idea at 30,6.MiB per second.

The SSL related libraries like PolarSSL, OpenSSL or Mozilla NSS perform well under the restrict group of ciphers: Rijndael AES, Triple DES (3DES) and Camellia but miss public-key schemes. Our asymmetric ciphers candidates for pSHIELD power node are 3 elliptic curve schemes: ECIES, ECDSA and ECDH.

In other hardware platform (other CPU, other motherboard) or with a Linux distribution, the best performance cryptographic libraries may change their relative positions. In our case, for the hardware under test, the best C/C++ libraries for the Power node are Beecrypt, Botan, Crypto++, Nettle and TomCrypt. The best java libraries are Bouncy Castle, FlexiProvider and Gnu-Crypt.

5 Conclusions

Deliverable D3.1 represents Node work package partners efforts to provide pSHIELD node layer solutions according to Technical Annex specification, and developed by consortium Requirements (D2.1.1), Metrics (D2.2.1) and Architecture (D2.3.1).

This is preliminary document presenting works in all WP3 tasks, covering wide rage of ES solutions including different kind of nodes like micro, nano/personal nodes and power nodes as well as security mechanisms used in that nodes. Complete description will be provided in final work package prototype reports D3.2, D3.3 and D3.4.

References

[1] D3.2 "SPD nano, micro/personal node technologies prototype report"

[2] D3.3 "SPD power node technologies prototype report"

[3] D3.4 "SPD self-x and cryptographic technologies prototype report"