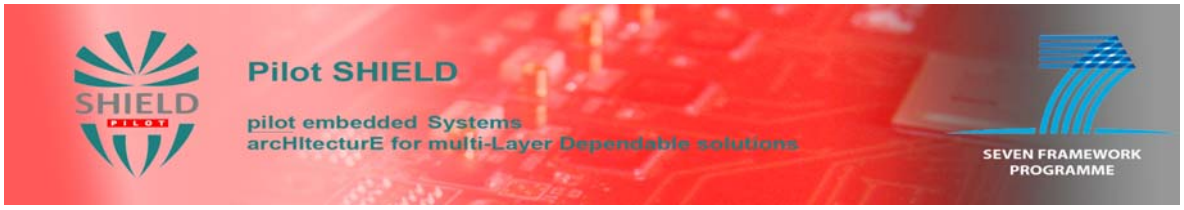


Project no: 100204

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 self-x and cryptographic technologies

Deliverable D3.4 Revision A

Partners that contributed to the work:

Acorde Seguridad, Spain
Critical Software, Portugal
ATHENA, Greece
THYIA, Slovenia

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

Document Authors and Approvals

Authors		**Date**	**Signature**
Name	Company		
Silvia Mier	AS		
José Veríssimo	CS		
Kiriakos Stefanidis	ATHENA		
Gordana Mijic	THYIA		
Nastja Kuzmin	THYIA		
Spase Drakul	THYIA		
Reviewed by			
Name	Company		
Approved by			
Name	Company		

Modification History

Issue	**Date**	**Description**
Draft A	1.09.2011	First issue for comments
Issue 1	20.12.2011	Incorporates comments from Draft A review
Issue 2		Incorporates comments from issue 1 review

Contents

Figures

Tables

Glossary

API	Application Programming Interface
AD	Applicable Document
AES	Advanced Encryption Standard
ANSI	American National Standards Institute
CA	certificate authority
CBC	Cipher Block Chaining
CC	Common Criteria for Information Technology Security Evaluation (ISO/IEC 15408)
CEM	Common Methodology for Information Technology Security Evaluation
CFB	Cipher Feedback
CMVP	Cryptographic Module Validation Program
CPS	Cyber-physical systems
CSW	Critical Software, S.A.
CTR	Counter
DES	Data Encryption Standard
DH	Diffie-Hellman
DHP	Diffie-Hellman Problem
DL	Discrete Logarithm
DLIES	Discrete Logarithm Integrated Encryption Scheme
DLP	Discrete Logarithm Problem
DPA	Differential Power Analysis
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
EAL	Evaluation assurance level
EBS	Exclusion-based systems

ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
ECDDHP	Elliptic Curve Decision Diffie-Hellman Problem
ECDH	Elliptic Curve Diffie-Hellman
ECDHP	Elliptic Curve Diffie-Hellman Problem
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
EC-KCDSA	Elliptic Curve Korean Certificate-based Digital Signature Algorithm
ECMQV	Elliptic Curve Menezes-Qu-Vanstone
EEA	Extended Euclidean Algorithm
EEPROM	Electronically-Erasable Programmable Read-Only Memory
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
EPROM	Erasable Programmable Read-Only Memory
ES	Embedded systems
FAU	Security audit FCS
FCC	Federal Communications Commission
FIPS	Federal Information Processing Standard
FNISA	French Network and Information Security Agency
FPGA	Field-Programmable Gate Array
GHS	Gaudry-Hess-Smart
GMR	Goldwasser-Micali-Rivest
HCDLP	Hyperelliptic Curve Discrete Logarithm Problem
HDL	Hardware Description Language

HMAC	Hash-Based Message Authentication Code
HSM	Hardware Security Module
IBM	International Business Machines
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IES	Integrated Encryption Scheme
IFP	Integer Factorization Problem
ISA	International Society of Automation
ISECOM	Institute for Security and Open Methodologies
ISO	International Organization for Standardization
ITSEC	Information Technology Security Evaluation Criteria
IV	Initialization Vector
JSF	Joint Sparse Form
KDF	key derivation function
KEM	Key Encapsulation Mechanism
LD	L□Lopez-Dahab
LEACH	Low-Energy Adaptive Clustering Hierarchy
LEAP	Localized Encryption and Authentication Protocol
MAC	Message authentication code
MANET	mobile ad hoc network
MD4	Message-Digest algorithm 4
MD5	Message-Digest algorithm 5
MEMS	Microelectromechanical systems
Mickey	Mutual Irregular Clocking KEYstream generator
NAF	Non-Adjacent Form

NEMS	Nanoelectromechanical systems
NESSIE	New European Schemes for Signatures, Integrity and Encryption
NFS	Number Field Sieve
NIKS	Non-Interactive Key Sharing.
NIST	National Institute of Standards and Technology
NSTISSAM	National Security Telecommunications and Information Systems Security Advisory Memorandum
OEF	Optimal Extension Field
OFB	Output Feedback
OSSTMM	Open Source Security Testing Methodology Manual
PGP	Pretty Good Privacy
PIN	Personal Identification Number
PKI	Public Key Infrastructure
PP	Protection Profile
PROM	Programmable Read-Only Memory
PSEC	Provably Secure Elliptic Curve encryption
pSHIELD	pilot embedded Systems arcHItecturE for multi-Layer Dependable solutions
RA	Registration Authority
RAM	Random Access Memory
RC4	Rivest Cipher 4
RD	Reference Document
RNG	Random Number Generator
ROM	Read-Only Memory
RSA	R. Rivest, A. Shamir e L. Adleman
SEC	Standards for Efficient Cryptography
SECG	Standards for Efficient Cryptography Group

SEED	Super Effective and Efficient Delivery
SHA	Secure Hash Algorithm
SIMD	Single-Instruction Multiple-Data
SOA	Service Oriented Architecture
SPA	Simple Power Analysis
SPD	Security, Privacy and Dependability
SPI	Serial Peripheral Interface
SPKI	Simple Public Key Infrastructure
SSL	Secure Sockets Layer
ST	Security Target
TBC	To be confirmed
TBD	To be defined
TESLA	Timed Efficient Stream Loss-Tolerant Authentication
TLS	Transport Layer Security
TTP	trusted third party
UCS	Use case Scenario
VLSI	Very Large Scale Integration
WEP	Wired Equivalent Privacy (IEEE 802.11)
WPA	Wi-Fi Protected Access
WSN	Wireless sensor networks
WWW	World Wide Web

This Page is intentionally left blank

1 Executive Summary

This document addresses automatic access control, denial-of-services (DoS), self-configuration and self-recovery technologies (self-x technologies). These technologies involve the adoption of proven methodologies against distributed DoS attacks and their applicability on a resource limited devices such as the SPD embedded systems (ESs) targeted in pSHIELD. Self-x technologies of sensing and processing tasks are proposed in order to guarantee robustness and dependability of the information collected from the SPD nodes described in D3.2 and D3.3. It represents a key feature at SHIELD's node layer and will also affect the performance of the overall pSHIELD framework, influencing SPD capabilities at network and middleware level. Self-x technologies will be provided for the SPD nodes by adopting field programmable gate arrays (FPGAs), programmable processor with a reconfigurable data path and a simple reprogrammable microcontroller. Hardware (HW) and Software (SW) crypto technologies are fundamental for achieving security of the SPD networks composed of SPD nodes.

One of targeted research topics is a study and design of embedded operating systems and firmware for energy-constrained SPD nodes. Choosing the right cryptographic technology for the three different ES Nodes is one of the most important research efforts dedicated in the design of SPD nodes and SPD network architecture. The research, validation and verification of different self-x technologies is the first step toward a right choice of the final system solutions for the pSHILD network composed of SPD nodes (nano, micro/personal and power). By selecting the best self-x candidate technologies they will be adopted in the SPD node technologies prototypes.

Today research effort is toward autonomous Wireless Sesnsor Networks (WSNs) with capable self-x technologies. We would like to create more versatile pSHIELD system by using adaptive SPD networks composed of hundreds of small nano, micro/personal sensor (NMPS) nodes, equipped with limited memory and multiple sensing capabilities. A design goal will be capability of the SPD nodes to autonomously organize and reorganize themselves as ad-hoc pSHIELD WSN network. The future SPD network architectures aim at solving some key problems of the standard static Internet architecture. To provide desired services an optimized way they have to adapt the SPD functionalities to different networking situations. This can be achieved by dividing the networking functionality into modular blocks and combining them as it is needed. Feasibility and flexibility of the SPD nodes was demonstrated in D3.2 and D3.3, and SPD network in D4.2. For providing desired performance on SW-based prototypes we are frequently limited due to lack of HW acceleration. Therefore, the aim of this chapter is to demonstrated feasibility of a networking SPD node architecture for the future generation of the pSHEILD networks to meet requirements for the future Internet applications. For that we need reconfigurable SPD node HW/SP platforms in which the SW modules of the network SPD protocol stack can be flexible and partitioned on HW and SW modules.

2 Introduction

The modern day embedded systems (ES) employ increasingly sophisticated communication technologies: low-end systems, such as wireless head-sets use standardised communication protocols to transmit data, remotely-controlled thermostats adjust room temperatures on user request sent from a mobile phone or from the Internet, while smart energy meters automatically communicate with utility providers. Furthermore, wireless sensor networks (WSN), or the recently emerging cyber-physical systems (CPS) are proposed to autonomously monitor and control safety-critical infrastructure such as, for example, a nation-wide power grid [1]. The increased complexity of these systems and their exposure to a wide range of potential attacks involving their communication interfaces makes security an extremely important and, at the same time, challenging problem. pSHIELD project recognizes the fact that security, privacy and dependability (SPD) are core characteristics of any modern ES and it proposes to address them as a “built-in” technology rather than as “add-ons”. In fact, due to the complexity of networked embedded systems, as well as because of the potentially high cost of failures, SPD must become an integral part of ES design and development [2].

3 Security in embedded systems, SPD nodes and networks

In the pSHIELD project the primary focus of SPD is addressed on the SPD nodes that are composing a SPD network. Therefore SPD issues considered in this document are related to securing the pSHIELD system. A **pSHIELD Node** is an Embedded System Device (ESD). When a Legacy ESD equipped with several legacy node capabilities will be used in the pSHIELD network it requires a pSHIELD Node Adapter (pSNA). A pSHIELD node is deployed as a hardware/software platform, encompassing intrinsic, innovative SPD functionalities, providing proper services to the other pSHIELD networks and middleware adapters to enable the pSHIELD composability and consequently the desired system SPD. There are three kinds of **pSHIELD node** each deploying different configuration's of Node Layer SPD functionalities of the pSHIELD framework, and comprising different types of complexity: **Nano nodes, Micro/Personal (NMP)** nodes and **power nodes**. Nano nodes are typically small ESD with limited hardware and software resources, such as wireless sensors. Micro/Personal nodes are richer in terms of hardware and software resources, network access capabilities, mobility, interfaces, sensing capabilities, etc. Power nodes offer high performance computing in one self-contained board offering data storage, networking, memory and multi-processing. While the three pSHIELD Node types cover a variety of different ESDs, offering different functionalities and SPD capabilities, they share the same conceptual model, enabling the pSHIELD seamless Composability.

The pSHIELD network architecture for the railway application scenario, the concept of four functional layers with SPD functionalities and core services is a **homogenous network** as in Figure 2.2 of the Technical Annex. By introducing more implicational scenarios as in nSHIELD and Legacy ES nodes and Legacy ES Networks, the final architecture becomes a **hybrid heterogeneous network** (HHN). It is heterogeneous in the sense of having coexistence different technologies (IEEE 802.15.4, IEEE 802.11, UMTS, etc., multi-frequency, multi- technology, multi-layer, multi-architecture) that are connected with unified control and optimisation, and it is a hybrid in the sense of a network that is between a centralised and pure decentralised architecture.

In D3.2 was demonstrated that the method for designing pSHIELD NMP Nodes is twofold:

1. To design completely **new NMP nodes** that are **complaint with the pSHIELD system** design.
2. To keep legacy node technologies as they are compliant with their standards, developed for many applications including those that are targeted in pSHIELD, which means to assume a heterogeneous infrastructure of networked ESDs like IEEE 802.15.4, IEEE 802.11, etc. An ordinary sensor technology (not all, since we need those that are designed for ES) permits to consider an augmentation of SPD functionalities at different levels of the hardware and firmware modules. This means an enhanced **legacy NMP node** with physical layer and protocol stack composed of existing and new SPD technologies added by pSNA. As result of this integration new types of networked SPD ESDs will be created. pSHIELD and new SPD ESDs will compose a heterogeneous SPD network infrastructure too.

Developing a NMP node as an integrated NMP-SPD Node of a Legacy NMP node and pSNA we obtain a composable pSHIELD Node. It means that it has all of the desired SPD functionalities and services for the pSHIELD application scenario selected. Additionally to that, the pSHIELD Node keeps some of the desired functionalities of a standardised sensor technology with **additional SPD features** that make it composable into the pSHIELD framework.

3.1 Securing embedded systems

Figure 3-1 illustrates the security pyramid[1] with five primary abstraction levels for an embedded system.

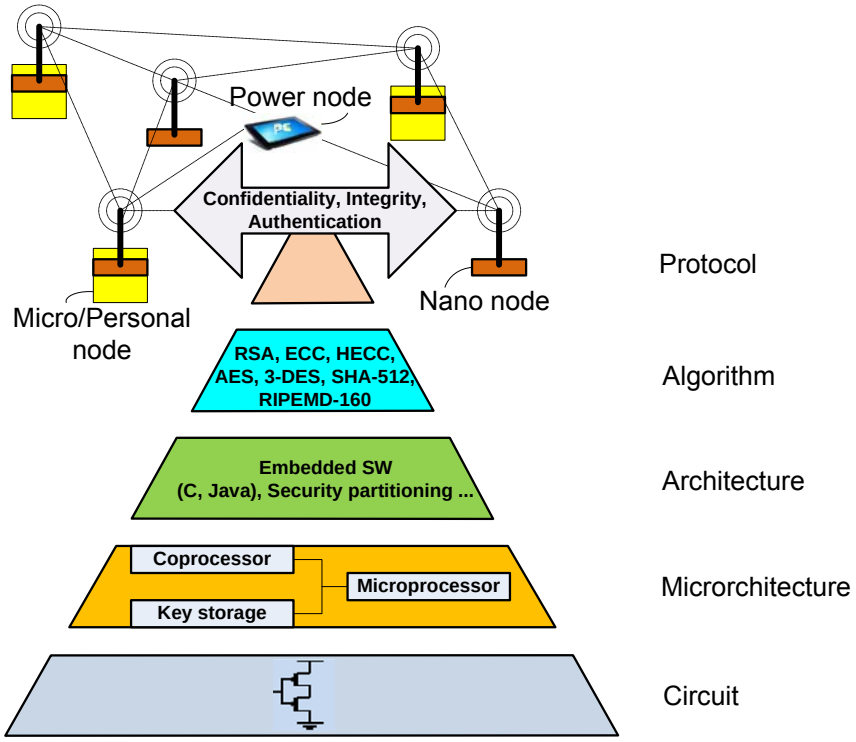


Figure 3-1: Embedded security pyramid.

The five abstraction levels are:

- **Protocol level** includes the protocols to be performed on embedded devices. For achieving security CIAA concept can be implemented.
- **Algorithm level** includes cryptographic primitives (such as Public Key, Symmetric Key crypto algorithms and hash functions) and application-specific algorithms used at the protocol level.
- ***Architecture* level** includes secure hardware/software partitioning and embedded software techniques to prevent software hacks.
- **Microarchitecture** deals with the hardware design of modules (the processors and coprocessors) required and specified at the architecture level.
- ***Circuit* level** requires implementing transistor level and package-level techniques to thwart various physical-layer attacks.

[1] D. Hwang, P. Schaumont, K. Tiri, and I. Verbauwhede. Securing Embedded Systems. In *IEEE Security and Privacy Magazine 4*, pages 40–49, 2006

3.1.1 Symmetric and asymmetric cryptography

Symmetric key based authentication (i.e. the claimer and the verifier share a key) is vulnerable to the compromise of either party in the authentication. In contrast, there is no secret key shared between the claimer and the verifier when using digital signatures. In public key cryptography (also called asymmetric key cryptography), a pair of keys including public key which is publicly available and private key which is kept as secret, are assigned to each entity. To authenticate to the verifier, the claimer signs a challenge message from verifier using its private key, and appends a digital certificate that confirms the link between the claimer and its public key. The verifier uses the certificate to verify the validity of the signer's public key and validates the integrity and authenticity of the message using the signer's public key. If an entity is no longer trustworthy, its certificate is revoked and the revocation is announced publicly by the certificate authority (CA).

Many implementations use **symmetric cryptography**, for example keyed hash functions or AES implementations, to meet the constraints of low-power consumption, limited chip area, and restricted computation time in order to produce low-cost devices. But in many application scenarios it is indispensable to obtain the high security level provided by an **asymmetric approach**. The use of asymmetric instead of symmetric solutions for different devices can radically reduce costs. Public-key approaches are more reasonable in open-loop applications, since no secret keys must be handled by the device. But the integration of public-key cryptography into low-cost devices is a technological challenge. Public key cryptography systems are usually based on the assumption that a particular mathematical operation is easy to do, but difficult to undo unless you know some particular secret. This particular secret serves as the secret key. A recent development in this field is the **elliptic curve cryptography** (ECC).

Protocol level is application specific and includes the design of protocols to be performed on EDs. The PKC (public key cryptosystems) are based on RSA[2] or DSA[3]. ECC (Elliptic Curve Cryptography) and Hyper-ECC (HECC) are based on different algebraic structure[4]. After 20 years of intensive investigation on both, theoretical and practical aspects it is evident that ECC and HECC offer equivalent security as RSA, but for much smaller key size! This result in smaller HW and lower power consumption that is extremely important for NMPS nodes. Modular multiplication forms the basis of modular exponentiation which is the core operation for RSA cryptosystems. Similarly, it is also important for ECCs especially if one use projective coordinates. Montgomery's methods[5] is the most popular for modular multiplication since it avoid time consuming trial division that is common bottleneck of other algorithms. However, it is not enough to have strong cryptographic algorithms. It is also important that their implementation that must be secured. The attack techniques are related to the PHY implementation. For example, the attack can be active or passive. Active attack is performed in such way to alter HW or SW by changing the operating conditions (power supply,

---

[2] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2):120–126, 1978.

[3] A. Menezes, P. van Oorschot, and S. Vanstone. Handbook of Applied Cryptography. CRC Press, 1997.

[4] V. Miller. Uses of elliptic curves in cryptography. In H. C. Williams, editor, Advances in Cryptology: Proceedings of CRYPTO'85, number 218 in Lecture Notes in Computer Science, pages 417–426. Springer-Verlag, 1985, and

N. Koblitz. Elliptic curve cryptosystem. Math. Comp., 48:203–209, 1987, and N. Koblitz. A family of Jacobians suitable for Discrete Log Cryptosystems. In S. Goldwasser, editor, Advances in Cryptology: Proceedings of CRYPTO'88, number 403 in Lecture Notes in Computer Science, pages 94–99. Springer-Verlag, 1988.

[5] P. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

temperature, etc.) Passive attack is based on monitoring side-channel information (power supply, EM radiation).

With this short introduction on pyramid security approach for ESs is clear that all abstraction levels must be secured. This is a complex security approach, which can be extended to a general pyramid SPD approach for the future ESs. For the pilot pSHIELD project we focus mainly on the security aspects for ESs.

3.1.2 General security requirements

The application area for pSHIELD is security monitoring RT of dangerous materials. Security monitoring WSNs are composed of NMP and power nodes (see D3.3) that are placed at fixed locations throughout an environment that continually monitor one or more sensors to detect an anomaly. Security WSNs are not actually collecting any data. For the environmental monitoring the collection of data is fundamental. Thus, for security WSNs for which data collection is not a primary goal, this has a significant impact on the optimal network architecture. Each node has to frequently check the status of its sensors but it only has to transmit a data report when there is a security violation. Once detected, a security violation must be communicated immediately to the power node and/or Gateway (GW) that is connected with the security control centre. The latency of the data communication across the WSNs to the GW has a critical impact on application performance. The end-users demand that alarm situations should be reported within seconds of detection. This means that NMP nodes must be able to respond quickly to requests from their neighbours to forward data. In security WSNs reducing the latency of an alarm transmission is significantly more important than reducing the energy cost of the transmissions. This is because alarm events are expected to be rare. When an event is present a significant amount of energy could be dedicated to the transmission. Reducing the transmission latency leads to higher energy consumption because routing nodes must monitor the radio channel more frequently. Therefore, in security WSNs, a vast majority of the energy will be spend on confirming the functionality of neighbouring nodes and in being prepared to instantly forward alarm announcements. Actual data transmission will consume a small fraction of the network energy.

The most important security requirements for WSNs are related to the CIAA (Confidentiality, Integrity, Authenticity and Availability) attributes. Measurable Security and Dependability attributes are integrity and availability. Immeasurable attributes are confidentiality and authenticity. For achieving secure communications for WSNs all messages have to be encrypted and authenticated. In the following sections we will briefly introduced the key security definitions and threats for WSNs.

3.1.2.1 Key security aspects for WSNs

3.1.2.1.1 Definitions

ACCESS CONTROL: Access control ensures that resources are only granted to those users who are entitled to them.

AUTHENTICATION: Verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system.

AVAILABILITY: Ensuring timely and reliable access to and use of information.

CONFIDENTIALITY: Preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information.

HELLO ATTACK: An attack in which an adversary attacks the network by repeatedly transmitting HELLO messages and thereby depletes the network's resources.

INTEGRITY: Guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity.

SECURITY OF INFORMATION: Appropriate technical and organizational measures to ensure an appropriate level of security in relation to the risks represented by the processing and the nature of the personal data to be protected.

SPOOFING: a spoofing attack is a situation in which one person or program successfully masquerades as another by falsifying data and thereby gaining an illegitimate advantage.

SYBIL ATTACK: An attack in which the attacker subverts the reputation system of a peer to peer network by creating a large number of pseudonymous entities, using them to gain a disproportionately large influence

THREAT: Any circumstance or event with the potential to adversely impact organizational operations, organizational assets, or individuals through an information system via unauthorized access, destruction, disclosure, modification of information, and/or denial of service.

VULNERABILITY: Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source.

WORMHOLE: A particularly severe attack on routing protocols in ad hoc networks, in which two or more colluding attackers record packets at one location, and tunnel them to another location for a replay at that remote location.

3.1.2.2 Typical threats in WSNs

The security requirements for pSHIELD WSNs are similar to those of wireless ad hoc networks due to their similarities[6],[7]. Communications over radio channels are in most of the cases insecure and easily susceptible to various kinds of threats. For WSNs composed of a high number of sensors, it is impractical to monitor and protect each individual sensor from physical or logical attack. Another approach is needed to cope with application that required a large scale WSNs (LS-WSN). Threats on WSNs can be classified into attacks on physical, link (MAC), network, transportation, and application layers. To perform good security architecture for pSHIELD system all OSI levels should be protected.

A powerful micro/personal and power node can do much more harm to pSHIELD system than a nano node, since it has much better power supply, as well as larger computation and communication capabilities than a simple nano sensor node. Threats can also be classified into outside and inside threats. An outside attacker has no access to most cryptographic materials in WSNs, while an inside attacker may have partial key materials and the trust of other pSHIELD sensor nodes. Inside attacks are much harder to detect and defend against. Typical threats and adequate defense techniques in WSNs are summarized as in Table 3-1 below[8].

---

[6] K. Lu e tal., » A framework for a Distributed Key Management Scheme in Heterogeneous Wireless Sensor Networks," IEEE Trans. on Wireless Communications, vol. 7, no. 2, Feb. 2008, pp. 639-647.

[7] Al-Sakib Khan Pathan e tal., "Security in Wireless Sensor Networks: Issues and Challenges," 2007.

[8] Z. S. Bojkovic et all, "Security Issues in Wireless Sensor Networks," International Journal of Communications, Issues 1, Volume 2, 2008, pp. 106-115.

Threats	OSI Layer	Defence techniques
Clone attack	Application	Unique pair-wise keys
Flooding	Transport	Limit connection numbers, client puzzles
Route inf. Manipulation	Network	Authentication, encryption
Selective forwarding		Redundancy, probing
Sybil attack		Authentication
Sinkhole		Authentication, monitoring, redundancy
Wormhole		Flexible routing, monitoring
Hallo flood		Two-way authentication, three-way handshake
Exhausting	Link/MAC	Rate limitation
Collision		Error correction code
Jamming	Physical	Spread-spectrum, lower duty cycle
Tampering		Tamper-proofing, effective key management schemes

Table 3-1: Typical threats in WSNs.

This table indicates what kind of countermeasure can be applied against some key threats that are typical for WSNs. Of course, approaches taking in consideration for the NMPS node prototypes are described in the next sections.

Taking in consideration those NMPS nodes will be with limited resources the access of these nodes in the field to perform SW updates can be difficult to locate or the nodes will be inaccessible. Performing remote update is associated with its own problems. Therefore, three key issues are important:

1. Avoiding interference
2. Minimizing the cost of upgrades
3. Avoiding the loss of part or all of a WSN

Figure 3-2 shows SW update model. There are three elements for SW updates with a flow from Generation (host), propagation (network) and activation (node). WSNs will be developed dynamically over time since in this field we have dramatically changes and improvements in the last five years. Standards and realises are upgraded with new features, algorithms and protocols that are evolving. This is an important driver for designing NMPS nodes and WSNs with such nodes that allow SW updates since this is a critical issues in the effective deployment of these networks as part of the pSHIELD network.

NMPS nodes, which are designed with five layers (PHY, link/MAC, network, transport and application) most of the important features and SPD functionalities will be included in the middleware to build up and maintain the network. For example routing, looking for the nodes, discovery services

and self-localisation. SW updates become important for many reasons: maintenance realises, minor realises, major realises, technology insertion, etc.

Basically for WSN we identified some fundamental design constraints such as security and reliability, routing and transport, and in-networking processing.

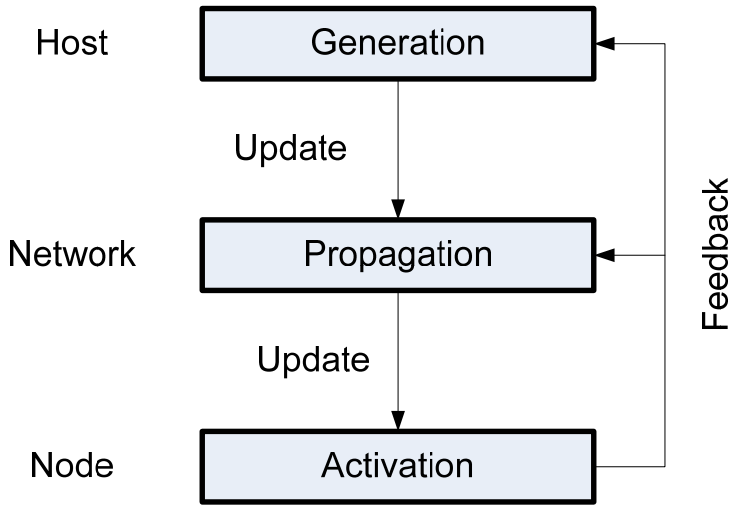


Figure 3-2: SW update model for WSNs.

First, the configuration of WSNs that are part of the pSHIELD network must be flexible enough to cope with abruptly disappearing nodes. The system must support routing and multiple levels of in-network processing. The second important issue is data aggregation. It is related to both the energy consumption at the sensor nodes and the effect of physical attacks on the node. To cope with this threat we need End-to-End (E2E) encryption from the NMP sensors to the sinks. This is also concerned for multicast traffic Concealed data aggregation provides a good balance between energy-efficiency and security. Data aggregation can be based on deviation query and multiple monitoring sensors. In some applications data must be stored in a distributed way. Since WSN is volatile with nodes that disappear over time, security must be combined with replication, taking space- and energy-efficiency storage. Resilient data aggregation at nodes addresses robustness against modified data. The solution must include both sensors and sink. Third, enhanced key pre-distribution is important because for the manufactures it is not possible to configure all the sensitive information before the WSN is deployed. Four, routing is one of the most important functionality of WSNs. The presence of malicious nodes must be taken in consideration and countermeasure must be applied for most of the known threats and attacks. Five, pairwise/groupwise authentication is important for establishing pair-wise relationships. For example, on top of CIAA security concept, Hop-by-Hop (HbyH) authentication offers strong security mechanisms. Finally, WSN access control is essential to provide an access control for end-users of WSN applications. This helps to monitor data for authorised users only.

3.1.3 Networked embedded systems

The current trends in ES design show a strong tendency towards the use of wireless communications, as well as of small, low-cost devices with sensing capabilities. The process started with the spread of mobile communications and, later, accelerated with the proliferation of local area wireless communication technologies such as Wireless LAN or Bluetooth. More recently, the development of low-cost, integrated wireless transceivers and MEMS sensors resulted in an explosion of research in the new field of wireless sensor networks.

Currently, wireless sensor networks are gradually making their way to the market promising near real-time monitoring of potentially large-scale areas [3]. Recent research proposes to extend distributed monitoring with actuation enabling this way distributed control of spatial processes and leading to a multitude of new applications that range from large-scale fire-prevention systems, through automated building energy management, to large-scale control of industrial systems and infrastructures. These technologies are often referred to as cyber-physical systems or wireless sensor-actuator networks (WSANs) and, although still in their infancy, they are widely expected to become dominant market drivers in the coming years [4].

These trends are further strengthened by the on-going standardization of wireless communication protocols for industrial applications such as, for example, Zigbee [5], ISA [6], Dash7 [7] or WirelessHART [8] and it is, therefore, reasonable to assume that the future embedded systems are likely to have at least some of the following characteristics:

- **Resource constraints:** Small, battery-operated wireless devices enable cheap sensing in hard-to-reach places and in harsh environments. The small-size factor and lack of cabling further increase the range of their possible applications in areas such as, for example, home appliances and consumer electronics. The advantages come, however, at a price of increased difficulty of software development and of securing the system due to the resource limitations, which usually take the form of small memory, low processing power and limited battery capacity.
- **Mobility:** Although fully autonomous systems may comprise only physically static devices, mobile network nodes might need to be used in many applications that require human interaction or supervision. These could take the form of personal data assistants (PDAs) or laptops and their presence adds to the complexity of securing the system since they may join and leave the network in different places in an unpredictable manner.
- **Heterogeneity:** The future embedded systems are likely to comprise devices of many different types. For example, a large scale monitoring and surveillance system could comprise different types of sensors such as, for example, digital cameras and passive infra-red (PIR) sensors, as well as data processing nodes and various actuators (e.g., remotely controlled door locks, sprinklers or alarms). Furthermore, industrial-grade distributed embedded systems might also require fixed infrastructure in the form of network routers, gateways and base stations. Security for heterogeneous embedded systems is challenging due to the fact that different parts of the system might have different computational capabilities, as well as different security requirements, thus precluding uniform application of the same security measures and techniques across the entire system.
- **Hierarchy:** Heterogeneous networked ES, especially when they comprise devices of radically different capabilities, often follow the hierarchical design pattern in which less capable devices are dependent on more powerful devices. This approach is a standard engineering practice in industrial control systems and has been recently suggested favourable for large-scale WSNs and WSANs in order to improve their overall energy efficiency and reliability [9].
- **Timeliness requirements:** Networked embedded systems that perform control tasks typically operate in a tight time regime, meaning that they need to execute control commands on time. Although the required degree of timeliness depends on the application, real time plays an important role in many ES and securing against timing-related attacks may prove difficult, as well as it is currently an active research topic [10].

All of these characteristics apply to the dependable surveillance system for urban railways, as described in the pSHIELD project's main application scenario. The system is envisioned to be a hierarchically-organised heterogeneous network of devices whose size and capabilities would span from large control room servers to small, battery-powered sensors.

3.1.4 Security Threats and Models

In section 3.1.1.2 are demonstrated standard threats in WSN. Here we will focus only on networked embedded systems. These systems are envisioned to perform tasks upon which human safety and prosperity might depend. For example, failures (either random or inflicted by an attacker) of a railway infrastructure-monitoring system might put the lives of train passengers in danger while flaws in the security of a distributed surveillance system might lead to noticeable financial losses. However, securing networked, heterogeneous embedded systems with potentially constrained resources is a challenging task. A distributed embedded system might have many users and complicated usage patterns resulting in sophisticated access control policies. Wireless communications, as well as physical distribution of system's components across potentially large areas significantly increase the diversity of possible attacks the system is exposed to. Finally, 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.

3.1.4.1 Attacks on Embedded Systems

There is a wide range of attacks that can be launched against embedded systems. The traditional Dolev-Yao [11] threat model focuses on the security of communication between two parties, in which each of which is considered to be secure and trusted (as a device). The model assumes that the attacker is able to overhear, intercept, capture and introduce its own messages to the communication channel and it is up to the communication protocol to ensure confidentiality, integrity and authenticity of the transmitted messages. However, although general and applicable to a large class of communication systems, the model is not well suited to embedded systems because the physical exposure of embedded devices to potential manipulation renders them untrusted.

3.1.4.1.1 Attacks on Cryptosystems

There are a number of techniques that have been used in the past to exploit weaknesses of some cryptographic algorithms and are currently used as basic evaluation criteria for new algorithms. The common aim of these attacks is to reveal partially or entirely the information encrypted in intercepted messages, or to extract some information internal to the encryption process (without initially knowing any secrets). They include:

- **Brute force attack:** traversing the entire encryption key space in order to learn the encryption key.
- **Dictionary attack:** related to the brute force attack in that a set of keywords are used as possible values of the encryption key (or a pass phrase).
- **Chosen cypher text attack:** obtaining information about a secret decryption key by submitting a range of cipher texts to decrypt. .
- **Adaptive chosen cypher text attack:** a version of chosen cypher text attack in which the attacker interactively selects subsequent cypher texts based on the results of decryption of the previous ones.
- **Cypher text-only attack:** the attacker has access to a limited set of cypher texts.
- **Known plain text attack:** the attacker has access to a number of cypher texts together with the corresponding plain texts.
- **Chosen plain text attack:** the attacker can encrypt an arbitrary set of chosen plain texts.
- **Adaptive chosen plain text attack:** like above, but the attacker chooses subsequent plain text for encryption based on the previous results.
- **Related-key attack:** the attacker has access to encryption of a plain text under several different keys whose exact values may not be known but which are somehow mathematically related.

In addition to these general attack methods, there is also a range of more general cryptanalytic techniques that may be used to study the properties of cyphers. They include frequency analysis, differential cryptanalysis, linear cryptanalysis, statistical cryptanalysis and mod-n cryptanalysis. Finally, there are also attacks on hashing functions (e.g., birthday attack) that aim at finding collisions in hash functions, or attacks on random number generators that exploit a generator's statistical weaknesses to simplify breaking a cipher that uses it.

3.1.4.1.2 Attacks on Protocols

Communication and security protocols can be attacked in a number of ways by intercepting and inserting messages in the communication channel. These attacks are even easier to perform in wireless networks since there might be little difficulty in accessing the channel, unless a more sophisticated technology such as direct-sequence spread spectrum (DSSS) or frequency hopping are used.

- **Replay attack:** resending of some captured messages in order to confuse the protocol or to exploit some of its weaknesses.
- **Wormhole attack:** a form of a replay attack that uses a low-latency and long-range transmission link to intercept communications in one part of the network and then to reproduce them in another network region, for example, with the goal of authenticating the attacker.
- **Man-in-the-middle attack:** the attacker intercepts all communications from a node A, modifies them and sends to a node B in such a way that both A and B have the illusion of direct communication with each other.
- **Bit flipping attack:** selectively flipping bits in intercepted messages in order to achieve desired protocol behaviour, for example, to route traffic to different recipients or to change the message type.
- **Attack on key distribution protocols:** preventing or intercepting key distribution in the network might severely affect the entire safety infrastructure of the system.
- **Routing protocol attacks:** the attacker may influence the contents of routing tables of some network nodes or even to introduce corrupt nodes to affect communication in the network.

3.1.4.1.3 Denial of Service

The main task of all embedded systems is to interact with the environment they are embedded in. Thus, there is a shift in the goals a potential attacker might want to achieve from simply trying to steal or forge confidential information, to also trying to prevent the system from achieving its design goals or even to deliberately damaging it. The denial of service (DoS) attacks may include the following:

- Physical damage.
- **Jamming of communication lines:** particularly important when wireless communications are employed.
- **System overloading:** the attacker may send a large number of requests making the system incapable of normal operation.
- Attacks on the system's power lines.
- **Battery depletion attacks:** the attacker may disrupt the operation of communication protocols with the goal of using up the remaining energy of battery-powered devices. For example, wireless sensor nodes are typically battery-powered and wireless transmissions consume a significant amount of energy. Engaging a node in continuous communications will quickly drain its batteries. Also, the attacker might try to circumvent the operation of a duty-cycling protocol in order to increase the network's duty cycles.

3.1.4.1.4 Physical and Side-Channel Attacks

Many modern cryptographic protocols are designed in such way that their security depends on the key rather than on the secrecy of the protocol's design. Thus, the security of an ES can be circumvented if the attacker has physical access to some of the system's components and is capable of extracting the keys. Depending on the capabilities of ES hardware and on the attacker's resources, there are many types of side-channel attacks that can be realised and they can be generally grouped in two categories: invasive and non-invasive. The former refers to the attacks that require physical tampering with a device, for example, micro-probing and reverse design engineering, while the latter comprises attacks that aim at extraction of cryptographic secrets through the analysis of the external effects of a device's operation.

Typically, tamper-proof hardware technologies are used in order to defend against invasive attacks. The main idea behind them is to be able to detect abnormal usage situations by means of specialized sensors or circuits and then to destroy the sensitive parts of the system, for example, by zeroing the key storage memory. Other techniques may include sealing for tamper evidence, tamper-proof casing, encryption of communication lines between hardware components (e.g., between the processor and the memory) or detection of abnormal clock rates and voltages.

The Federal Information Processing Standard FIPS 140-2 [12] defines four levels of physical security requirements for secure embedded systems: Level 1 devices have minimum physical protection, Level 2 devices implement tamper evidence mechanisms such as seals or enclosures, Level 3 devices implement both tamper evidence and tamper response mechanisms (e.g., sensitive memory zeroing) and Level 4 devices must implement all the requirements for Level 3 devices, as well as they must provide for environmental failure protection and response mechanisms. Thus, it is clear that there is a spectrum of options available for physical ES security and, typically, the desired security level is traded off against the financial cost of implementing it.

On the other hand, non-invasive side-channel attacks do not require devices to be physically accessed, thus they are usually easier and cheaper to perform. They include the following techniques:

- **Timing analysis:** the attacker might be able to infer the keys of an encryption algorithm by measuring small variations in the time the device needs to perform cryptographic computations.
- **Power analysis:** similarly to the timing attack, power analysis aims at the extraction of the device's secret information by precise measurements of the variations in the devices power consumption (the current draw) throughout the execution of its cryptographic algorithm.
- **Electromagnetic analysis:** extraction of security information through the analysis of the device's electromagnetic radiation. For example, [13] demonstrate that it is possible to reliably read the sequence of keystrokes from a neighbouring room by using a high-grade antenna.

3.1.4.1.5 Attacks on Control Systems

Control systems are a category of embedded systems that operate in a close connection with a real-world process such as, for example, a flight control system or a chemical plant controller. Usually, these systems make their control decisions in real time and might be vulnerable to timeliness inaccuracies the effects of which may sometimes be catastrophic. Following [10], the possible attacks might include:

- **Time synchronization attack:** the attacker can influence a distributed time synchronization protocol in order to desynchronize different parts of the system. Also, hardware manipulation or system overloading might be used to achieve the same effect.

- **Attack on sensors:** the attacker may influence responses of the system's sensors thus destabilising its control loop.

3.1.4.2 Attacker Model

Security engineering is a discipline in which one seeks to provide measures of dealing with the unexpected and security systems can only protect against an a priori known set of threats. Thus, in order to define the scope of the system, one needs to define the broadest possible set of attacks the system might be exposed to.

The most typical assumptions in the security literature correspond to the security of the communication links. We should assume, therefore, the following capacity of the attacker with respect to the messages transmitted by network nodes:

- Reception
- Delay
- Insertion
- Reordering
- Corruption
- Deletion

These assumptions encompass a wide range of possible attacks: deletion of messages can be used to model jamming of communication lines in wireless networks, while safety of such network protocols as key distribution, routing and duty cycling necessarily depends on the attacker's ability to modify message contents in an authentic way.

Another group of assumptions, crucial for the domain of ES, is the ability of the attacker to tamper with the network nodes. These, however, depend on the hardware technologies employed and on the expected resources of the attacker. There is already a large body of research on tamper-proof hardware and there is also a range of commercial products available (for an overview see [14]). Because none of these technologies offer 100 per cent security, their choice should necessarily involve matching their strength against their cost, the required security level and the expected resources of a potential attacker. For example, a wireless network of low-cost temperature sensors used in home environment could, perhaps, use less sophisticated tamper-proof technology, but a system responsible for surveillance of critical infrastructure should use the best technology available since the potential gain from corrupting it might attract attackers with substantial resources.

Since the tamper-proof security levels may vary between different embedded systems and they change over time (upgrades are needed as new types of attacks are devised), it is reasonable to assume that the attacker can extract secret information from all types of devices, but with a varying probability of success. As a result, for every type of technology used, an estimate can be made on the time needed for an expected attacker to compromise the device. This approach makes the provision of system's security more realistic by enforcing distribution of security responsibilities among other system components.

Finally, it has to be assumed that the attacker is able to command the compromised devices towards his goals and, thus, it has the ability to damage the system from the inside, unless proper counter measures (such as, for example, intrusion detection and separation of privileges) are implemented.

3.1.5 Security Requirements

This section discusses what should be expected from the security infrastructure in order to build a dependable ES. The basic requirements of any security system are typically concerned with the security of communication links and thus must include the following:

- **Authentication:** the ability to assure the identities of parties participating in a protocol. In the context of embedded systems, authentication typically means the ability to distinguish between the original and forged data packets and devices.
- **Integrity:** the ability to state whether communication messages have been tampered with. Integrity and authentication allow assuring both the origin and the contents of messages.
- **Confidentiality:** the ability to conceal the contents of communication messages. Confidentiality is, in general, orthogonal to authentication and integrity although encryption can be used to implement integrity.

These three basic requirements need to be extended in order to satisfy the threats ES can be exposed to. One of the key requirements for the security and dependability of an ES is graceful degradation of its services under random or inflicted (by an attacker) faults. A networked ES that comprises many nodes should gradually degrade the quality of its outputs as subsequent devices are either taken under the attacker's control or simply fail. This means, in particular, that after capturing one or few devices, the attacker cannot obtain access to the rest of the network, for example, by extracting the global network encryption key or by commanding the compromised devices to bring the network down. Instead, the damage has to be local, i.e., only the compromised devices and, perhaps, some of their network neighbours should be considered faulty. Also, as a consequence of the ability of the attacker to eventually compromise any device in the system, it has to be assumed that none of the network nodes can be fully trusted and measures need to be taken to ensure detection of abnormal node behaviour (an intrusion detection system should be in place).

If wireless sensor networks are to be a part of the system, then WSN-specific security requirements should be considered. In particular, the SPD security infrastructure should not preclude the ability of the sensor nodes to listen to their neighbours' traffic, to process data packets while they are routed towards their destination, as well as it should not preclude duty cycling.

Finally, we consider the following issues to be relevant to networked ES and, thus, require counter-measures to be implemented by the SPD middleware:

- Time synchronization is likely to be an integral part of any networked ES, thus it has to be implemented in a secure way.
- The capacity of the attacker to delete arbitrary messages already encompasses a DoS attack that is based on jamming communication lines. It is not sufficient, however, to simply conclude that an encryption or an authentication protocol should be able to handle such situation because this would render the system useless and unable to perform its main objectives. Therefore, robust wireless communications together with such techniques as channel and node black-listing should be implemented.
- Denial-of-service attacks that aim at battery depletion and system overloading should be prevented.
- The routing protocol used should be robust to maliciously-behaving compromised nodes.

3.1.6 Design Challenges

Meeting the presented set of requirements is a challenging task due to a number of reasons of which limitations in hardware capabilities and the available bandwidth play a central role. According to [2], there is a gap between the requirements of the available security protocols and the capabilities of the existing ES architectures. High grade cryptography exceeds the capabilities of small embedded processors and remains costly when applied to high-rate traffic on high-end servers. Furthermore, energy consumption overheads of security technologies are significant and, in the case of small battery-powered devices, they may become prohibitive.

Tamper-resistance technologies are absolutely necessary for ES applications because these systems often operate unattended and are thus exposed to a greater range of security risks than

home or office PCs. In fact, since system's security is measured by the security of its weakest element, over-investing in cryptographic protocols and algorithms without due attention to its physical exposures might prove pointless. However, striking a perfect balance between the sufficient level of physical security and the cost of the system might be very difficult and it requires careful definition of the system's security goals.

Another issue is the difficulty of implementing security updates on networked embedded systems. Security systems are never perfect and updates are often required due to implementation faults, design flaws or new types of attacks being discovered. Updating an ES might prove difficult for a number of reasons. The system might comprise numerous devices, already deployed in places that potentially are hard to access, or it might be infeasible to switch off the system due to its importance. Finally, implementing a remote update functionality adds to its complexity and it might be a great source of vulnerabilities on its own.

Finally, security assurance, i.e., the probability that the system's security goals have been met, might be difficult to assert, especially in the case of real-time networked ES since their complexity and the range of security exposures grows significantly. Theoretical properties of security protocols matter only as long as their implementations are faithful. This issue is also closely related to the problem of security updates.

4 Self-reconfigurability and self-adaptation of sensing and processing tasks

Today research effort is toward autonomous WSNs. We would like to create more versatile pSHIELD system by using adaptive SPD networks composed of hundreds of small NMPS nodes, equipped with limited memory and multiple sensing capabilities. A design goal will be the capability of the SPD nodes to autonomously organize and reorganize themselves as ad-hoc pSHIELD WSN network. The future SPD network architectures aim at solving some key problems of the standard static Internet architecture. To provide desired services an optimized way they have to adapt the SPD functionalities to different networking situations. This can be achieved by dividing the networking functionality into modular blocks and combining them as it is needed. Feasibility and flexibility of the SPD nodes was demonstrated in D3.2 and D3.3, and SPD network in D4.2. For providing desired performance on SW-based prototypes we are frequently limited due to lack of HW acceleration. Therefore, the aim of this chapter is to demonstrate the feasibility of a networking SPD node architecture for the future generation of the pSHIELD networks to meet requirements for the future Internet applications. For that we need reconfigurable SPD node HW/SP platforms in which the SW modules of the network SPD protocol stack can be flexible and partitioned on HW and SW modules.

4.1 Comparison of static and adaptive network architecture

Figure 4-1 illustrates a standard protocol stack of the WSN and a adaptive architecture that between PHY and the application layer is a adaptive middleware composed of SW modules.

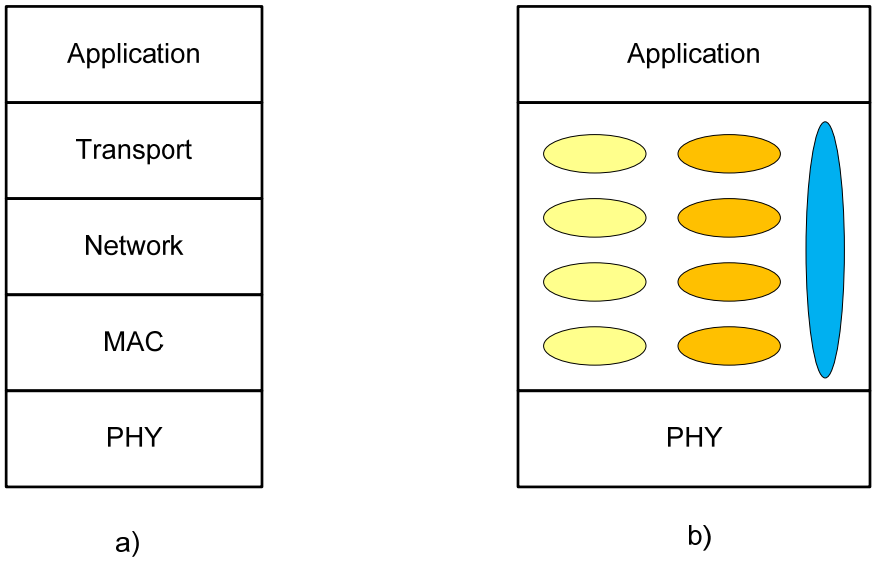


Figure 4-1: a) The protocol stack of a standard WSN (e.g. composed of IEEE 802.15.4 type of sensor nodes); b) an adaptive protocol stack composed of SW modules.

An ASIC implementation is infeasible since we need to know all functions precisely at design time. The new architecture will adapt their functionality to the current run-time communication needs. In order to combine the processing power of an ASIC implementation with the flexibility of a SW implantation a FPGA implementation can be a good choice. Even a hybrid platform combined of a microcontroller and a FPGA unit can be a good candidate for SPD node architecture. It can provide an additional measure of flexibility. However, FPGA offers a realization of programmable logic capable of implementing high-performance data-parallel HW in a reconfigurable SoC.

4.1.1 An adaptive middleware layer

The sensor node for standardised WSNs differs so much in terms of HW platforms. Writing an OS that runs on all these sensor platforms is impossible. To hide the underlying platform differences and to decouple the OS from HW platform a middleware is needed. The concept of middleware in distributed systems is often taken to mean the software layer that lies between the operating system and the applications on each site of the system. It facilitates scalability, interoperability, deployment, and development of applications.

In the last decade numerous works on middleware for mobile devices (smart phones) are performed and successfully implemented. Most of those devices use operating systems like Windows CE, Palm OS, Symbian OS, Tiny Linux, etc. But in this document we focus on middleware for NMP nodes, which are much smaller than those devices. The recent development of sensor node middleware is showing that we have quite a large number of middleware for WSNs. Most of the middleware we have studied are built on top of TinyOS. There are other OSs like Contiki, Mantis, SOS, and t-kernel.

It is important to note that the scope of middleware for WSN is not restricted to the sensor network alone, but also covers external networks connected to the WSN (such as Internet) as well as the applications interested in querying sensor data through such external network. Standards such as 6LoWPAN (which used IEEE 802.15.4) and Web Services running directly on the sensor node allow integrating them into the Internet of Things (IoT). However, nodes which are capable to run the internet stack directly are either very expensive or not very energy-efficient. There have been several efforts to implement the Internet Protocol Stack on small energy-constrained devices. The LoWPAN and 6LoWPAN protocols try to port the IPv4 and IPv6 Protocols on small devices. This enables running services on the application layer directly on sensor nodes. The Web service technology is often used to connect and access sensors and actuators through the Internet.

The recent middleware approaches use different technique. For example, such middleware are Sensorpedia (Web 2.0 based), TinyDB (Database oriented), Mate (Virtual Machine based), Agilla (Mobile Agent), TinyLime (tuple space) and TinyCubus (cross-layered).

Taking in consideration that pSHIELD SPD network is composed of SPD and Legacy Nodes it is obvious that we have a complex HHN structure where the standard OSI layers are defining the overall network requirements in sense of the HW & SW components. On the physical layer (PHY) different NMP nodes will coexist in the same pSHIELD network. Above PHY different protocol stacks for different Legacy NMP nodes are increasing the complexity of the overall pSHIELD network design. Figure 4-2 illustrates a standard Internet Layer Structure and Middleware for WSNs composed of SW components that adapt the PHY layer to the application layer.

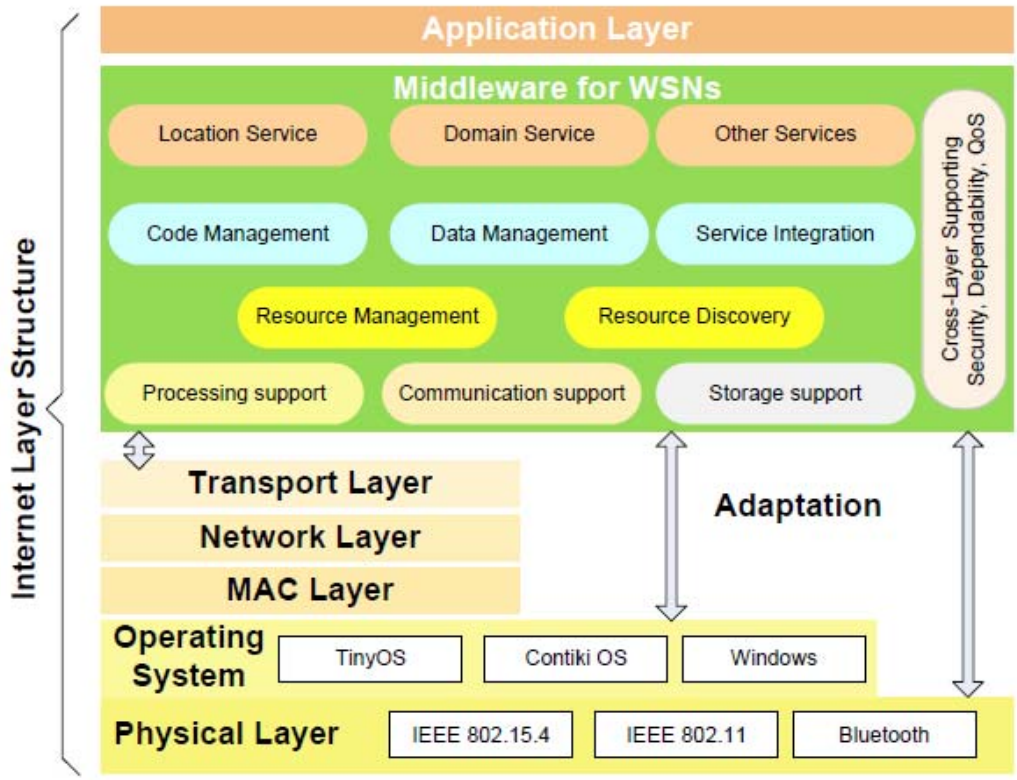


Figure 4-2: A reference model for middleware in WSNs.

Based on the requirements for NMPS nodes for WSNs and the main challenges in the development of adaptable middleware (as it is shown in Figure 4-2) a base for comparison of the following features is:

- Code Mobility: it evaluates the support of code mobility, both for update and installation of new services in a node;
- Flexibility: it evaluates the support for network scalability and support for manage incoming nodes in the network, as well as manage for the network topology;
- Node Mobility: it evaluates the support for mobile nodes in the network;
- Node Heterogeneity; it evaluates the capacity of the middleware address the needs of both low-end nodes, with few and constrained resources, as well as more sophisticated sensors, with more powerful computer platforms and advanced resources;
- Application Knowledge: it evaluates the ability of the middleware to respond the needs of specific applications or group of applications;
- Data Fusion: it evaluates the support for data aggregation and fusion by the nodes that are in the way of data moving from the phenomenon occurrence to the end user that requested the information;
- QoS: it evaluates the support for QoS control that can be provided by the middleware.

By comparing some adaptable middlewares[9] like DAVIM, ATLAS, AGILLA, IMPALA, SINA, TinyCubs, MiLAN, SensorWare, TinyLime, and AWARE, the conclusion that can be drawn from this analysis is that there is a need to integrate the support for each of the described feature a common middleware platform in order to offer the required support for new emerging applications. There have been also some efforts to architect middleware for WSNs using SOA (RUNES, P2PComp, etc). SOA can deal with aspects of heterogeneity, mobility and adaptation, and offers seamless integration of wired and wireless environments.

The OSGi (Open Services Gateway Initiative) is focused on the application layer. It is open to almost any protocol, transport or device layers. The OSGi mission is multiple services, wide area networks, and local networks and devices. The OSGi advantages are platform and application independent. The central component of the OSGi specification effort is the services gateway. Service semantics for WSNs is another important issue, in addition to the service definition, so that services can be coordinated in the space.

Hydra platform[10] is a new concept that is realised in such a way that between physical and application layer is a middleware. The main goal was to develop a middleware that is 'inclusive' which means that it will be possible to enable any device to be detectable and usable from a Hydra application. The concept is based on the work of Rozanski and Woods[11] , and the Hydra architectural descriptions are in line with the IEEE 1471 standard. For the NMP prototype platform design concept we will explain in the following section how it can be composed by an operating system, middleware and the application layer.

The European Hydra project developed a "Middleware for Heterogeneous Physical Devices" with the aim to help manufacturers and systems integrators to build devices that can be networked easily and flexibly to create cost-effective high performance solutions. For the heterogeneous devices, sensors and actuators envisioned in the pSHIELD project, the large number of manufacturers and Universities are involved and the differences in their speed of innovation become an obstacle for the overall system design. Therefore, there is an urgent need for technologies and tools that make it easier to reap the benefits of networked systems. The complexity to build new technologies and tools grows exponentially with the number of devices, manufacturers and protocols involved.

The Hydra middleware as in Figure 4-3 is a core technology that has a transparent communication layer, equally supporting centralised and distributed architectures. The Hydra middleware takes security and trust into account and allows building model-guided web services. It runs on wired or wireless networks of distributed devices with limited resources. The embedded and mobile service-oriented architecture will provide fully compatible data access across heterogeneous platforms, allowing true ambient intelligence for networked ESDs. Adding extended security, privacy, trust and new dependability modules may satisfy requirements for having a middleware that will be SPD composable with the rest of the pSHIELD system architecture and network. The Hydra middleware consists of large number of software components – or managers – that handle various tasks needed to support cost-effective development of intelligent applications for networked embedded devices.

[9] Pignaton de Freitas, “A Survey for Adaptable Middleware for Wireless Sensor Networks, Technical Report, 2008.

[10] http://www.hydramiddleware.eu/news.php

[11] Rozanski, N. and Woods, E. , “Software systems architecture: working with stakeholders using viewpoints and perspectives,” Pearson Education.

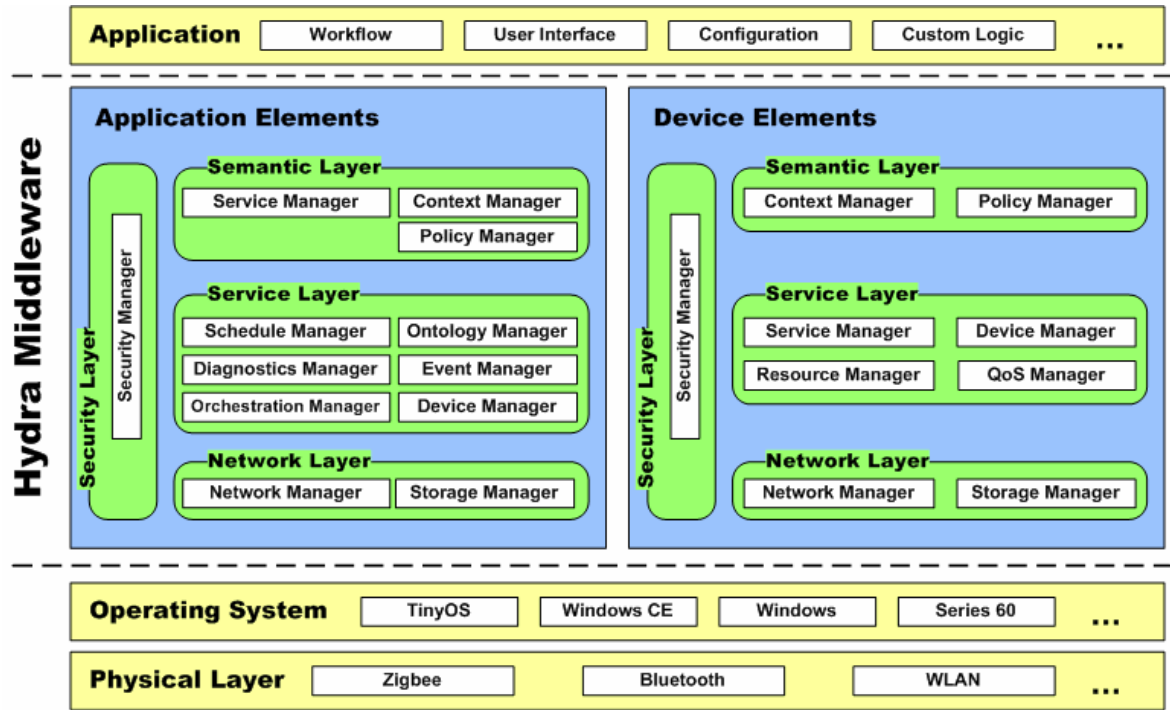


Figure 4-3: The Hydra middleware layer[12].

The biggest advantage of the Hydra middleware relies on the fact that allows developers to incorporate heterogeneous ESDs into their applications. This middleware can be incorporated in new and existing networks of distributed ESDs, which operate with limited resources: computing power, energy and memory. Additionally, Hydra-middleware provides easy-to-use web service interfaces for controlling any type of physical device irrespective of its network interface technology. Additionally, this middleware is based on a semantic Model Driven Architecture for easy programming and incorporate service discovery, P2P communications and diagnostic. In Hydra framework any physical devices, sensor, actuators or subsystem can be considered as a unique web service.

What we will need from Hydra middleware for the pSHIELD SPD nodes? A lightweight version of this middleware, to be the Legacy Middleware Layer on top of which the pSHIELD middleware Adapter can host a set of Innovative SPD Functionalities: proper software modules must be added. This solution is in line with the recent IP stacks that are lightweight enough to run on tiny, battery operated ESDs. This is also in line with emerging application space of smart objects that require scalable and interoperable communication mechanisms that support future innovations as the application space grows. This strategy is also aligned with the future application scenarios the "Internet of Things and Human" (ITH). Smart objects are small computers with a sensor and actuator and a communication device, embedded in objects. To support the large number of emerging applications for smart objects, the underlying networking technology must be inherently scalable, interoperable, and have solid standardization base to support future innovation.

[12] Hydra project , D3.4, "Initial architectural design specification," http://www.hydramiddleware.eu/articles.php?article_id=90

4.1.2 A reconfigurable NMP-SPD node

The concepts developed in D3.2 for NMP-SPD nodes will be now extended toward an adaptive NMPS node architecture. It can be implemented on a reconfigurable SoC node combined of a microcontroller and an FPGA as in Figure 4-2.

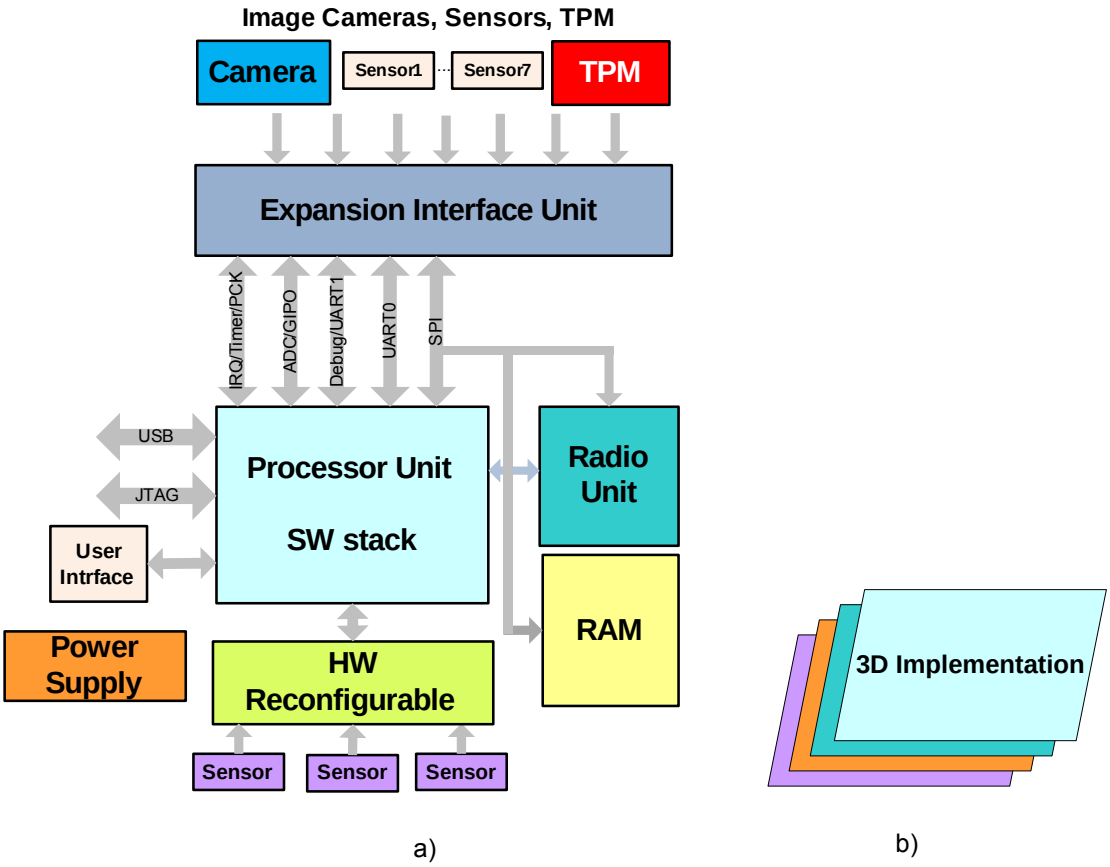


Figure 4-4: a) A reconfigurable NMP-SPD node architecture, b) 3D implementation.

A reconfigurable NMP-SPD node can be designed as a prototype in micro-3D integration. The node will have the following board layers as in Figure 4-4 b).

- Microcontroller unit
- Radio unit
- Power supply
- Sensors
- TPM
- Memory
- HW reconfigurable unit

Additionally, to the standard layers of NMP-SPD node other layers can be added in 3D integrated platform, for example memory a HW reconfigurable unit. This architecture permits to split analog and digital sensors. Analog sensors (1 to 7) are connected by expanding connector to ADC of the microcontroller. Signals from digital sensors are connected on the FPGA. FPGA is responsible for digital sensors to realize management of node communications and processing data from analog sensors as well as other SPD functionalities. The new generation of sensor are equipped with

different protocols such as SPI, I2C, etc, which allow a direct connection to the microcontroller. However, when these signal have to be processed using a microcontroller problems related to timing and processor overhead can appear. The FPGA unit will overcome these problems. For example the sensor connected to FPGA may be responsible for sensing different SPD functionality on the node network, and application layers. The microcontroller sends signal to FPGA specifying the sensor from which the SPD metric has been taken, and the FPGA activities the appropriate sensor interface. Then, the FPGA sends the results to the microcontroller. Therefore, the FPGA acts as a reconfigurable coprocessor for the microcontroller by taking signals from the digital sesnors., and processing the information for the microcontroller.

5 Hardware and Software crypto technologies

5.1 Embedded OS and firmware

5.1.1 NMP node operating systems

Selection of the operating system (OS) for the demonstrator is an important design constraint, since we need to decide in which sensor prototype platform will be realised SPD functionalities. The only requirement that we posed for this operating system is related to it's possibility to be designed for embedded devices. There are two candidates for that: TinyOS and Contiki

5.1.1.1 Node operation systems

5.1.1.1.1 TinyOS

This operating system (OS) is a free and open source operating system and platform that is designed for WSNs. It is an embedded operating system, written in the nesC Programming language as a set of cooperating tasks and processes. NesC is actually a dialect of the C programming language that is optimised for the memory limitation of the sensor networks. TinyOS features summary:

- No Kernel: Direct hardware manipulation.
- No Process Management: Only one process on the fly.
- No Virtual Memory: Single linear physical address space.
- No S/w Signal or Exception: Function call instead.
- No User Interface, power constrained.
- Unusually application specific H/w and S/w.
- Multiple flows, concurrency intensive bursts.
- Extremely passive vigilance (power saving).
- Tightly coupled with the application.
- Simulator: TOSSIM, PowerTOSSIM
- Written in "nesC" Language, a dialect of the 'C' language.

5.1.1.1.2 Contiki Operating System

Contiki is also an open source, highly portable, multi-tasking operating system for memory-efficient networked ESDs and WSNs. It is mainly designed for a microcontroller with small amount of memory. The key advantage of Contiki OS is its IP communications (both IPv4 and IPv6). It is flexible for a choice between full IP networking and low-power radio communication mechanisms. Contiki is written in the C programming language and consists of an event-driven kernel, on top of which application programs can be dynamically loaded and unloaded at run time. Contiki has been ported to different hardware platforms, such as MSP430, AVR, HC 12, and Z80. Contiki features summary:

- Event-driven Kernel: reduce the size of the system.
- Pre-emptive multi-threading support: an application library that runs on top of the event-driven kernel is optionally linked with applications that explicitly require a multithreaded model of computation.
- Simulator: COOJA
- Written in 'C' Language.

5.2 Cryptographic technologies

5.2.1 Digital Signatures

Digital Signatures have been designed in order to provide the digital counterpart to a handwritten signature. A digital signature is a number that depends on some secret only known to the signer (the signer's secret key) and the content of the message to be signed. The design goal is to make the signature verifiable, i.e. an unbiased third party should be able to check, without knowing the secret of the signer, whether the message has been indeed signed by a particular person. Such verification may be necessary when either the signer denies having signed the message (repudiation) or when an adversary has faked a signature and claims that it is valid.

5.2.1.1 The RSA Signature Scheme

The RSA signature scheme was discovered by Rivest, Shamir, and Adleman [15]. It was the first practical signature scheme based on public-key techniques. The integer factorization problem (IFP) is the underlying computationally hard mathematical problem for the RSA signature scheme and is the following problem:

Find p and q, if you are given a composite number n that is the product of two large prime numbers p and q.

Since it is part of the public key, an adversary has access to the modulus n. Once p and q are computed, the system is broken and the attacker can determine the secret key.

Algorithm 1 summarizes how a key pair, i.e. a public and the corresponding private key, for the RSA signature scheme can be generated. The steps for signing a message are given in The Algorithm 2 shows the steps for signing a message and Algorithm 3 shows how verifying a message.

Algorithm 1: Key generation for the RSA signature scheme

OUTPUT: The public key (n, e) and the private key d.

1: Generate two large distinct random primes p and q, each roughly the same size.
2: Compute $n = pq$ and $\phi = (p-1)(q-1)$.
3: Use the extended Euclidean algorithm to compute the unique integer d, $1 < d < \phi$, such that $ed \equiv 1 \pmod{\phi}$.
4: The public key is (n, e); the private key is d.

Algorithm 2: RSA Signature generation

INPUT: The message m, the public key (n, e) and the private key d.

OUTPUT: The digital signature s.

1: Compute the hash value of the message $\tilde{m} = h(m)$, an integer in the range $[0, n-1]$.
2: Compute $s = \tilde{m}^d \bmod n$.
3: The signature for m is s.

Algorithm 3: RSA signature verification

OUTPUT: The message m, the public key (n, e) of the signer and the signature s on m.

1: Compute $\hat{m} = s^e \bmod n$.
2: Verify that $\hat{m} = h(m)$; if not, reject the signature.

Attacks for the IFP:

- Continued Fraction Algorithm: This algorithm is based on the idea of using a factor base of primes and generating an associated set of linear equations whose solution leads to a factorization. It could factor numbers of up to 133-bits.
- Quadratic Sieve Algorithm (QS): This algorithm is based on the same idea as the continued fraction algorithm and can be easily parallelized to permit factoring on distributed networks.
- General Number Field Sieve (NFS): Also based on the idea of the continued fraction algorithm, the NFS is supposed to be the fastest known algorithm for factoring integers having at least 400 bits.
- Elliptic Curve Factoring Method (ECM): This algorithm attempts to exploit special features of an integer to be factorized. It tends to find small factors first.

5.2.2 The Digital Signature Algorithm (DSA)

The Digital Signature Algorithm has been proposed in August of 1991 by the U.S. National Institute of Standards and Technology (NIST). The underlying computationally hard mathematical problem is the discrete logarithm problem (DLP):

> Given a prime p, a generator α of $\mathbb{Z}_p$, and a non-zero element $\beta \in \mathbb{Z}_p$, find the unique integer $l, 0 \leq l \leq p-2$, such that $\beta = \alpha^l \pmod p$.

The integer l is called the discrete logarithm of β to the base α. If p is a prime number, then $\mathbb{Z}_p$ is a finite field denoted by the set of integers $\{0, 1, 2, \ldots, p-1\}$, where addition and multiplication are performed modulo p. There exists a non-zero element $\alpha \in \mathbb{Z}_p$ such that each non-zero element in $\mathbb{Z}_p$ can be written as a power of α; such an element α is called a generator of $\mathbb{Z}_p$.

Algorithm 4 summarizes how a key pair, i.e. a public and the corresponding private key, for the DSA signature scheme can be generated. The steps for signing a message are given in Algorithm 5 and the steps for verifying a message can be found in Algorithm 6.

Algorithm 4: Key generation for the DSA

OUTPUT: The public key (p, q, α, y) and the private key a.

1: Select a 160-bit prime q and a 1024-bit prime p with the property that $q \mid p-1$.
2: Select a generator α of the unique cyclic group of order q in $\mathbb{Z}_p^*$, i.e. select an element $g \in \mathbb{Z}_p^*$, compute $\alpha = g^{(p-1)/q} \bmod p$, an repeat this if $\alpha = 1$.
3: Select a random integer a such that $1 \leq a \leq q-1$.
4: Compute $y = \alpha^a \bmod p$.
5: The public key is (p, q, α, y); the private key is a.

Algorithm 5: DSA signature generation

INPUT: The message m, the public key (p, q, α, y) and the private key a.

OUTPUT: The digital signature (r, s).

1: Select a random integer $k, 0 < k < q$.
2: Compute $r = (a^k \bmod p) \bmod q$.
3: Compute $k^{-1} \bmod q$.
4: Compute $s = k^{-1}\{h(m) + ar\} \bmod q$, where $h(m)$ is the hash value of the message m.
5: The signature for m is (r, s).

Algorithm 6: DSA signature verification

INPUT: The message m, the public key (p, q, a, y) of the signer and the signature (r, s) on m.

1: Verify that $0 \leq r \leq q$ and $0 \leq s \leq q$; if not, reject the signature.
2: Compute $w = s^{-1} \bmod q$ and the hash value $h(m)$.
3: Compute $u_1 = w * h(m) \bmod q$ and $u_2 = rw \bmod q$.
4: Compute $v = (a^{u_1} y^{u_2} \bmod p) \bmod q$.
5: Accept the signature if and only if $v = r$.

According to [105], there exist the following known attacks against DSA and the discrete logarithm problem:

- Index Calculus Method: The fastest general-purpose algorithms known for solving the DLP are based on the index calculus method. In this method, a database of small primes and their corresponding logarithms is constructed. Subsequently, logarithms of arbitrary field elements can be easily obtained. This is reminiscent of the factor base methods for integer factorization. If an improvement in the algorithms for either the IFP or DLP is found, then shortly after this a similar improved algorithm can be expected to be found for the other problem. The index calculus method can be easily parallelized.

- Number Field Sieve Algorithm: This is the best current algorithm known for the DLP and has precisely the same asymptotic running time as the corresponding algorithm for factoring integers.

5.2.3 The Elliptic Curve Digital Signature Algorithm (ECDSA)

A variant of DSA based on elliptic curves is ECDSA. It was first proposed in 1992 by Scott Vanstone. Signature algorithm is used for authenticating a device or a message sent by the device. For example consider two devices A and B. To authenticate a message sent by A, the device A signs the message using its private key. The device A sends the message and the signature to the device B. This signature can be verified only by using the public key of device A. Since the device B knows A's public key, it can verify whether the message is indeed send by A or not.

The underlying computationally hard mathematical problem is the Elliptic Curve Discrete Logarithm Problem (ECDLP):

> Given an elliptic curve E defined over $\mathbb{F}_q$, a point $P \in E(\mathbb{F}_q)$ of order n, and a point $Q \in E(\mathbb{F}_q)$, determine the integer l, $0 \leq l \leq n-1$, such that $Q = lP$, provided that such an integer exists.

This discrete logarithm problem over elliptic curves is considered to be significantly harder than the DLP over $\mathbb{Z}_p$, which is the mathematical basis for DSA. Therefore, the strength per-key-bit is substantially higher than in DSA and, hence, smaller parameters (keys) can be used for elliptic curve cryptosystems to achieve equivalent levels of security.

In order to facilitate interoperability, the domain parameters for ECDSA, which are the parameters of the curve E, the underlying finite field $\mathbb{F}_q$ and a base point $G \in E(\mathbb{F}_q)$, have to be negotiated and agreed upon by the communication partners. The curve is usually determined by its two parameters a and b and the curve equation. For the finite field $\mathbb{F}_{2^m}$, the curve equation is given by the equation

$$y^2 + xy = x^3 + ax^2 + b,$$

which is the same for all m. The base point G is defined by its affine coordinates x_G and y_G. Usually, the order n of the point G is also part of the domain parameters.

Algorithm 7 summarizes how a key pair, i.e. a public and the corresponding private key, for the ECDSA signature scheme can be generated. The steps for signing a message are given in Algorithm 8 and the steps for verifying a message can be found in Algorithm 9. Signature generation and verification requires the computation of the hash value of the message using the Secure Hash Algorithm (SHA-1), which was proposed by the U.S. National Institute for Standards and Technology (NIST).

Algorithm 7: Key generation for the ECDSA

INPUT: The elliptic curve domain parameters.

OUTPUT: The public key Q and the private key d.

1: Select a random integer d in the interval $[0, n-1]$.
2: Compute $Q = dG$.
3: The public key is Q and the private key is d.

Algorithm 8: ECDSA signature generation

INPUT: The message m, the elliptic curve domain parameters, the public key Q, and the private key d.

OUTPUT: The digital signature (r, s).

1: Select a random integer $k, 0 < k < n$.
2: Compute $kG = (x_1, y_1)$ and convert x_1 to an integer.
3: Compute $r = x_1 \bmod n$. If $r = 0$ then go to Step 1.
4: Compute $k^{-1} \bmod n$.
5: Compute SHA-1(m) and convert this bit string to an integer e.
6: Compute $s = k^{-1}(e + dr) \bmod n$. If $s = 0$ then go to Step 1.
7: The signature for the message m is (r, s).

Algorithm 9: ECDSA signature verification

INPUT: The elliptic curve domain parameters, the message m, the public key Q of the signer and the signature (r, s).

1: Verify that r and s are integers in the interval $[0, n-1]$.
2: Compute SHA-1(m) and convert this bit string to an integer e.
3: Compute $w = s^{-1} \bmod n$.
4: Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.

5: Compute $X = u_1 G + u_2 Q$
6: Compute $X = O$, then reject the signature. Otherwise, convert the x-coordinate of X to an integer x_1, and compute $v = x_1 \bmod n$.
7: Accept the signature if and only if $v = r$.

5.2.4 Cryptographic Algorithms

Cryptography has been a vital means to information protection over the years. Cryptographic techniques stretching back to ancient Egyptians and beyond have evolved with the advancement of maths. Ultimately, cryptography is seen as the basis for the provision of different systems security, fundamentally by seeking to achieve a number of goals, that are; confidentiality, authenticity, data integrity and non-repudiation. Typically, 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 becomes vital. This section covers different asymmetric and symmetric cyphering algorithms as well as key management techniques. Often, cryptographic algorithms are presented side by side with the key management techniques, however, we try to differentiate between these in this document. Understanding that the pSHIELD project is highly concerned with lightweight devices, we discuss the suitability of the studied technologies accordingly.

5.2.4.1 Asymmetric Cryptography

Asymmetric cryptography, also known as public key cryptography [15], 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 [16]; 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.

5.2.4.1.1 Rivest Shamir Adleman

RSA [15] 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 latter is mainly concerned with finding the e^{th} roots of an offhand number modulo n. This n is a composite integer that results from the product of two distinct large prime numbers p and q. In other words, how to efficiently find M when only one is aware of the congruence relation $C \equiv M^e \pmod n$ and the public key (n, e) where C is the cyphered text. Typically, the public key is randomly selected from the range $1 < e < ((p-1)(q-1))$, such that $gcd(e, (p-1)(q-1)) = 1$. Also, the private key d is computed based on the range $1 < d < ((p-1)(q-1))$, such that the congruence relation $ed \equiv 1 \pmod{((p-1)(q-1))}$ is satisfied. Naturally, to decrypt a message C in order to obtain M, the following is computed, $M = C^d \pmod n$.

RSA on its own cannot be considered safe given the algebraic nature of the algorithm. For example, the plaintext message relation $M1\ M2 = M3$ is preserved after the encryption process among the cyphered output, i.e., $C1\ C2 = C3$. This is known as the holomorphic nature of raw RSA. However,

the use of hashing and padding techniques in addition to random nonce on plaintext helps in solving that problem.

RSA has been successful in applications deployed on resourceful infrastructures. However, its applicability on constrained device and infrastructures is not evident normally due to the large key size required and the computational complexity incurred. For example, the RSA key size recommended for usage beyond 2010 should be 2048 bits as opposed to ECC's 224 bits [17].

5.2.4.1.1.1 Implementations

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 [18] as well as software implementations [19]. 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) [20] 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 [21] where an open source implementation is provided in GnuPG [22].

The suitability of RSA's implementations in the light of constrained devices is discussed later on in this document.

5.2.4.1.2 Elliptic Curve Cryptography

ECC [23] 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 [17]. 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.

5.2.4.1.2.1 Implementations

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 [16]. For example, ECC hardware-based cyphers are recommended for the security needed in authenticating smart cards as they require less information exchange [17]. 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 favourable. 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.

5.2.4.1.3 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 [24], 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 [25].

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 $\mathbb{Z}^*p$. 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 $m \in \mathbb{Z}^*p$ 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.

5.2.4.1.3.1 Implementations

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 [26].

5.2.4.1.4 Suitability 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) [27], the American National Standards Institute (ANSI) [28], the Standards for Efficient Cryptography Group (SECG) [29] and the National Institute of Standards and Technology (NIST) [30], have all adopted ECC for lightweight devices [16]. In an energy analysis study comparing ECC and RSA implementations for a similar but slower hardware [17], 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 hardware used for the analyses was the Berkley/Crossbow motes. 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. However, it may not be a complete solution.

5.2.4.2 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 finalists of these competitions with the goal of making a recommendation for the pSHIELD project.

5.2.4.2.1 Advanced Encryption Standard

In January 1997, the National Institute of Standards and Technology of the United States (NIST) [30] organized a competition in order to 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. Although improvements were proposed, such as, for example, the Triple-DES, performance considerations increased the need for a new and efficient symmetric cipher that would offer stronger security guarantees. Fifteen candidate ciphers were submitted to the competition and by April 2000 five finalists were chosen. In 2001, Rijndael was selected for the new encryption standard and, since then, it is referred to as AES. In the following sections, the finalists are presented and we start the presentation with DES because of its historical significance.

5.2.4.2.1.1 DES

DES is a block cipher with 64 bit keys (of which only 56 bits are actually used by the algorithm) that operates on 64 bit blocks of plaintext. It is based on a balanced Feistel network, which is a cryptographic construction pioneered by the German-born cryptographer Horst Feistel in his work on the Lucipher cipher. Fiestel networks have good cryptographic properties and a large proportion of block ciphers proposed to date use them.

DES was selected in 1976 by the National Bureau of Standards of the United States to be an official Federal Information Processing Standard (FIPS) [33] for unclassified government communications. Since then, the cipher was subject to substantial academic scrutiny and its design greatly influenced subsequent developments.

The cipher's main criticism focussed on its relatively small key space, which gave way to the design of a range of brute force attacks. In 1997 RSA Security organised a series of competitions in decryption of DES messages. The contest ended 1998 when the Electronic Frontier Foundation demonstrated the ability to crack the cipher in 22 days using custom-built hardware that cost $250.000.

These results led to a number of modifications being proposed in order to improve the cipher's shortcomings. The most notable were DESX and Triple DES (3DES). The former uses two additional 64 bit keys of which one is XOR-ed with the plaintext prior to encryption (so called, key whitening) while the other is XOR-ed with the resulting cipher text. Although the total nominal key length is 184 bits, the effective total key length is approximately 120 bits due to a number of vulnerabilities of the scheme.

Triple DES, on the other hand, is a modification in which the original DES algorithm is applied three times to the plaintext, each time with a different key. Similarly to DESX, the effective key length was declared lower than the combined nominal length of 168 bits (three times 56 bits) to be only 80 bits due to certain meet-in-the-middle and chosen cipher text attacks. Despite these shortcomings, Triple DES is believed to be secure and has many applications, for example, in the electronic payment industry.

5.2.4.2.1.2 Rijndael

Designed by two Belgian cryptographers, Joan Daemen, and Vincent Rijmen, Rijndael is the winner of the AES competition. Since then, 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].

5.2.4.2.1.3 Serpent

The cipher came as the second in the AES contest and is widely regarded as a more secure but slower counterpart of Rijndael. Serpent is a 32-round substitution-permutation network and uses the same parameters as all other AES submissions, i.e., the block size is 128 bits and the keys can be either 128, 192 or 256 bits long. It was designed with hardware parallelization in mind and its structure allows for high degree of parallel execution.

The cipher takes a conservative approach to security by specifying double the rounds necessary for it to be secure in order to defend against unknown future attacks. To date, there has been no realistic attacks proposed on its full-round version and the most powerful attack presented involves 10 out of the 32 encryption rounds.

Serpent was reported to perform encryption and decryption approximately two to four times slower, and to perform its key setup approximately three times longer than Rijndael [31]. Although the cipher's designers state that Serpent can be faster than DES [37], the AES contest performance comparison [36] observes that the cipher is considerably slower than a good DES implementation (at the time DES was regarded fast and it was considered to be the point of reference for performance evaluation of all AES submissions). Finally, the AES contest jury declared that "the best Serpent software implementation will be 3-4 times slower than the other candidates" [36].

5.2.4.2.1.4 Twofish

The cipher is based on a Feistel network and uses key-dependent S-boxes, as well as a relatively complex key schedule. Twofish uses half of every key's bits for the actual encryption and decryption while the remaining half is used in the definition of the S-boxes, the structure of which is computed during the key setup procedure.

Twofish is based on the design of an older and more established cipher Blowfish, which was originally developed in 1993 and which was intended as an early replacement for DES. Blowfish gained popularity because of its security characteristics, as well as because it was one of the few publicly available block ciphers at the time. As of 2010, both ciphers are considered secure and no realistic cryptanalysis has been found to compromise any of them. However, since Blowfish is more established and has received more public attention and scrutiny, its security record is somehow stronger.

Twofish was originally optimized for 32-bit CPUs and it can encrypt and decrypt data fast on different hardware platforms, offering performance similar to that of Rijndael [31]. However, the key setup procedure has significantly higher complexity (its computational cost is roughly equivalent to encryption of 4kB of data) and, thus it makes the cipher more suitable for encryption of data streams and larger files, rather than of sporadic network traffic such as, for example, detection reports of a WSN.

Both Twofish and Blowfish are not patented and are completely in the public domain. Blowfish is used in more than two hundred software products and Twofish, although less popular due to its relatively recent introduction, is gaining acceptance through its use in such systems as GnuPG [22] and PGP [38].

5.2.4.2.1.5 RC6

The cipher was designed by RSA Security, a company that was originally founded by the inventors of the RSA asymmetric encryption algorithm, and which also holds a patent on it. RC6 is based on RC5, which is another cipher designed and patented by RSA Security. The cipher was submitted to the three main standardization contests: the AES, the European NESSIE project [39] and the Japanese Government's CRYPTREC project.

RC6 is very similar in its design to RC5 and it can be viewed as interweaving of two parallel RC5 encryption processes. The cipher follows the Feistel's approach encryption and its main design goal was to improve its predecessor's security parameters. The original submission of the cipher to the AES contest uses the standard block size of 128 bits, key lengths of 128, 192 or 256 bits, and it uses twenty encryption rounds. However, similarly to RC5, the cipher has a more general design and it admits a wide variety of block sizes, key lengths, as well as different numbers of rounds. The use of

twenty encryption rounds is believed to provide a good security margin and linear cryptanalysis, as well as it was reported to be effective only up to 16 rounds, while differential cryptanalysis theoretically can be used to break up to 12 rounds [40].

The main performance limitation of RC6 is its reliance on specialised hardware support for multiplication and rotation that is not available on many CPUs, in particular, on RISC and low-end processors. Thus, there is a considerable variety in the cipher's performance results across different hardware platforms. On the other hand, RC6 has favourable memory requirements that make the cipher suitable for implementation on limited-resource devices such as smartcards (although Rijdael still fares better in this respect).

5.2.4.2.1.6 MARS

MARS was submitted to the AES contest by IBM and it is a cipher that uses three-phase layered encryption process whose core consists of a 16-round unbalanced Feistel network. The cipher has 128-bit block size and uses key sizes that can vary between 128 and 448 bits in 32-bit increments. The full encryption process comprises thirty two rounds and this value, according to the authors, provides sufficient security margin to defend against future attacks.

The cipher was reported to have weaknesses with respect to certain classes of keys, as well as it turns out to be possible to recover two least significant bits of round keys [41]. Also, the cipher's complex design makes its security assessment more difficult.

MARS is a relatively fast cipher but its performance varies across different hardware platforms and compilers. Its main drawback is its use of ROM: the cipher needs at least 2KB of memory just to store its S-boxes and the full implementation was estimated to require at least 3KB of ROM, thus making it difficult to implement the cipher on memory-constrained devices without specialized hardware support.

5.2.4.2.2 NESSIE

The NESSIE project (New European Schemes for Signatures, Integrity and Encryption) was a European Union initiative to identify secure cryptographic primitives and it can be regarded as a European counterpart for the Advanced Encryption Standard. The project was organised in 2000-2003 and it accepted submissions under four main categories: block ciphers, public-key encryption, MAC algorithms and cryptographic hash functions, digital signature algorithms and identification schemes. The final recommendation of the project excluded stream ciphers because all submissions fell under cryptanalysis (this fact led to a separate call for stream ciphers that was announced as the eSTREAM project, which explicitly aimed at finding reliable and robust stream ciphers).

The project selected the following block ciphers for its portfolio: MISTY1, Camellia, SHACAL-2, and AES. The ciphers are presented in more detail in subsequent sections.

5.2.4.2.2.1 MISTY1

MISTY1 is a block cipher with 64-bit blocks that was designed for Mitsubishi electric and which was also recommended for government use by the Japanese CRYPTREC project. The cipher uses 128-bit keys and it is a Feistel network with a variable number of rounds (eight rounds are recommended for security). The innovative feature of the design is the use of non-linear, invertible FL functions. The cipher is patented but it is available royalty-free for non-commercial applications.

The cipher had been studied for over five years and no significant security threats had been found [42]. Attacks on a simplified version of MISTY1 succeeded only up to five out of the eight recommended rounds. The project report states that the use of FL functions significantly improves

the cipher's resistance to a range of different cryptanalysis. However, the use of 64-bit keys makes it potentially susceptible to brute-force attacks, as it was demonstrated in the case of DES.

MISTY1 can be implemented on heavily resource-constrained devices and its general encryption performance is bit better than that of DES [31]. Its design makes it also suitable for hardware implementations.

5.2.4.2.2.2 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].

5.2.4.2.2.3 SHACAL-2

The cipher is based on the cryptographic hash function SHA-2, which was introduced by NIST in 2000. The cipher uses relatively large block size of 256 bits (it can also be configured to use 512-bit blocks) and it supports key sizes of 512 bits. Its design differs from the other ciphers presented so far in that it uses a hash function as the core bit mixing transformation, as well as that the cipher's security depends on the security of the hash function used (this was the main vulnerability of the related cipher SHACAL-1 that used the SHA-1 hash function and which was reported to be insecure).

To date, no realistic attack on SHACAL-2 has been found [42] and the strongest attack proposed to date involved only 42 out of the cipher's 80 rounds.

The cipher is relatively fast and its encryption performance was reported to be slightly better than that of Camellia, although the key setup takes more time [31]. The larger block and key sizes, however, make the cipher more suitable for encryption of data streams and larger files rather than aperiodic radio messages, as in the case of networked embedded systems.

5.2.4.2.3 CRYPTREC

Cryptography Research and Evaluation Committees (CRYPTREC) was a project organised by the Japanese government to evaluate and monitor the security of e-Government recommended ciphers. In the year 2000, a call was made for the following categories of cryptographic primitives: public-key algorithms, symmetric-key ciphers, cryptographic hash algorithms and cryptographic pseudo-random number generators. In February 2003, the final selection was made and the following symmetric ciphers were proposed:

- **64-bit blocks:** CIPHERUNICORN-E, Hierocrypt-L1, MISTY1, Triple DES.
- **128-bit blocks:** AES, Camellia, CIPHERUNICORN-A, Hierocrypt-3, SC2000.
- **Stream ciphers:** MUGI, MULTI-S01, RC4 (128-bit keys only).

The subsequent sections present those ciphers that have not already been discussed. Also, the SEED block cipher is presented because of its wide use in South Korea and because it was used as an evaluation target by the CRYPTREC project.

5.2.4.2.3.1 CIPHERUNICORN

CIPHERUNICORN-E and CIPHERUNICORN-A are block ciphers designed by NEC in 2000. Both ciphers are 16-round Feistel networks of which the former has a simpler structure than the latter. The round function is split, in both cases, in two nearly parallel Feistel sub-networks and it has a significant complexity making the ciphers difficult to analyse. Contrary to its predecessor, CIPHERUNICORN-A is more general allowing for key sizes of 128, 192 and 256 bits.

There is not much information available on the security of the cipher but the analysis done by the CRYPTREC panel suggests that is resistant to linear and differential cryptanalysis [44], [45].

CIPHERUNICORN-A was reported to be a relatively slow cipher. It can encrypt and decrypt data at approximately 53 Mbps and this places the cipher on par with Triple DES while being five times slower than Camellia [46].

5.2.4.2.3.2 Hierocrypt

Hierocrypt-L1 and Hierocrypt-3 are two block ciphers that were designed by Toshiba in 2000 and which were submitted to both the CRYPTREC and NESSIE projects. Both ciphers are substitution-permutation networks that are similar in their designs differing mainly in key and block sizes and the numbers of rounds used. Hierocrypt-L1 operates on 64-bit blocks of plaintext and 128-bit keys, while Hierocrypt-3 uses 128-bit blocks and it supports keys of 128, 192 and 256 bits. Also, the former uses 6.5 encryption rounds while the latter uses 6.5, 7.5 and 8.5 for the different key sizes respectively.

Although the CRYPTREC project recommends the use of the cipher in governmental applications, the NESSIE project panel rejected both streams on the grounds that attacks were found that significantly reduced the ciphers' security margins [42].

Both Hierocrypt ciphers exhibit fairly good encryption performance faring at approximately 75% of that of Camellia. However, the cipher has very costly key setup procedure hence it is not suitable for low-power networked embedded systems.

5.2.4.2.3.3 SC2000

Similarly to Hierocrypt, SC2000 is a cipher that was submitted to both the CRYPTREC and NESSIE projects and which was recommended by the former and rejected by the latter. The cipher was designed by Fujitsu Labs in 2000 and it is a combination of a Feistel network and of a substitution-permutation network [47]. The cipher operates on 128-bit blocks and uses 128, 192 and 256-bit keys.

No cryptanalysis of the full-round cipher has been found so far but there is a differential attack on 4.5 out of the original 6.5 rounds of the 128-bit version of the cipher [48], [49]. There is a boomerang and rectangle attack on the reduced version [50].

SC2000 is relatively efficient cipher but it performance varies across platforms. In particular, data encryption on legacy platforms (e.g., the 486 processor) can be almost four times slower than Camellia. Also, the cipher has a slow key setup [31].

5.2.4.2.3.4 MUGI

MUGI is a random number generator intended to be used as a stream cipher. MUGI was designed by Hitachi Ltd. in 2001 and submitted to the CRYPTREC project [51]. The cipher uses 128-bit keys and 128-bit initialization vectors (IV) and it outputs 64-bit strings. MUGI has a 1216-bit internal state and its design reuses the definition of non-linear S-boxes, originally defined by Rijndael, as well as, it reuses the Rijndeal's MDS matrix (used for diffusion purposes).

MUGI's design is particularly well suited for hardware implementation since it can be implemented with relatively small number of gates. For example, an implementation with 26 thousand gates is capable of encryption rate of 2.922 Gbps [52].

Although no practical attacks on the cipher's security have been found to date, a number of theoretical weaknesses have been identified. In particular, the design of the linear part of the cipher, i.e., of its buffer, may theoretically facilitate linear cryptanalysis [53]. Additional weakness where found in [54] and [55].

5.2.4.2.3.5 MULTI-SO1

MULTI-S01 is a stream cipher that also implements message authentication code (MAC). The cipher was developed by Hitachi Ltd. In 2000 its design builds upon the Panama random number generator (the design of which influenced also the design of MUGI). The cipher uses 256-bit keys and 256-bit initialization vectors, as well as it feeds in plaintexts in 64-bit words.

The security of the cipher strongly depends on the security of the Panama random number generator [56], which at the time of analysis was considered a secure primitive. Since then, Panama was shown to be susceptible to a practical attack when used as a hash function. Finally, the CRYPTREC committee rated MULTI-S01 among the fastest ciphers submitted.

5.2.4.2.3.6 RC4

RC4 is a widely used stream cipher that was designed by RSA Security and which was originally a trade secret until its design was leaked in 1994. Since then the cipher became public although the company never officially acknowledged releasing it. RC4 is renowned for the simplicity of its design and for the efficiency of its software implementation and it has been used in popular protocols such as SSL or WEP.

The algorithm uses variable-length keys that may range from 40 to 128 bits. Contrary to more recent designs, the cipher does not require provision of an initialization vector (i.e., a random nonce) leaving it up to the implementation to introduce randomization to the encryption process in order to avoid security risks that are related with key reuse.

There have been numerous attacks on the security of RC4 in over the last decade. Correlations have been found in the cipher's key stream and the most effective attack on the 104-bit version used in the WEP protocol achieved breaking the protocol in less than a minute [57]. Currently, RC4 is not recommended for use in new systems.

The algorithm has very favourable performance characteristics. It is suitable for efficient implementation in both software and hardware and it has small memory footprint. Indeed, these were the characteristics that led to the widespread adoption of the cipher.

5.2.4.2.3.7 SEED

SEED (Super Effective and Efficient Delivery) is a block cipher developed in 1998 by the Korean Information Security Agency. The cipher is a 16-round Feistel network that operates on 128-bit data blocks and uses 128-bit keys [58]. The cipher is widely used in South Korea but it is not common outside of the country.

The cipher has a good security record meaning that no serious attacks have been found in spite of the analysis by the international panels of ISO/IEC [59] and of the CRYPTREC project. In addition to the ISO/IEC standardization, the cipher is also used by the IPsec protocol [60].

The cipher can be efficiently implemented both in software and hardware and its relatively small memory requirements make it feasible to implement the cipher on such memory-constrained devices as smartcards. The NESSIE encryption performance evaluation places SEED between Camellia and Serpent, although it has to be noted that the cipher's key setup procedure is faster than that of AES [31].

5.2.4.2.4 eSTREAM

The eSTREAM project (2004-2008) was funded with the aim of selecting a set of reliable stream ciphers following the failure of all stream ciphers submitted to the NESSIE project to resist cryptanalysis. The submitted ciphers were divided into two main categories that corresponded to the suitability of the ciphers for implementation either in hardware or in software. The project recommended the following stream ciphers:

- Implementation in software: HC-128, Rabbit, Salsa20/12, Sosemanuk.
- Implementation in hardware: Grain v1, Mickey v2, Trivium.

The ciphers are briefly discussed in the following sections.

5.2.4.2.4.1 HC-128

HC-128 and HC-256 are two freely available stream ciphers designed by Hongjun Wu for submission to the eSTREAM project. HC-256 is an extension of HC128 that uses 256-bit keys and initialization vectors instead 128 bits. The streams were designed to demonstrate that use nonlinear feedback and nonlinear output functions can be used to build strong and efficient stream ciphers. Both streams are considered secure and no effective cryptanalysis has been found to date.

Both ciphers are very fast and HC-128 was the fastest stream cipher among the secure one submitted to the project. However, Kircanski and Amr Youssef presented an DFA attack that requires about 7968 faults and recovers the complete internal state of HC-128 [61]. An cache timing attack on HC-256 is presented by Zenner [62]. Other observations are in [63].

The initialization procedure is costly and the cipher is expected to perform poorly in packetized communications, which are typical to low-power wireless networks [64], [65].

5.2.4.2.4.2 Rabbit

The cipher is among the oldest designs submitted to the project and so far no significant cryptanalytic attacks have been found giving it a strong security record. The cipher uses 128-bit keys and 64-bit initialization vectors but in some scenarios the effective key length has been shown to be reducible to 96 bits.

The cipher is very efficient when implemented in software but its design gives also way to efficient and small hardware implementations [66], [65]. However, it has a relatively costly key setup procedure.

In some scenarios with a large number of keys, the key strength is reduced to 96 bits [67](Y. Lu, H. Wang, e Ling 2008). See also [68], [69] and the DFA presented by Kircanski and Youssef [70].

Rabbit was patented but since 2008 is available royalty-free for general use.

5.2.4.2.4.3 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].

5.2.4.2.4.4 Sosemanuk

Sosemanuk is a software-oriented stream cipher whose structure was influenced on the SNOW 2.0 and Serpent ciphers. The cipher's key length may vary between 128 and 256 bits and it uses initialization vectors of 128 bits. The cipher offers a considerable security margin although a vulnerability was found for key lengths longer than 226 bits. The cipher is not patented and it is free for any use.

The cipher was demonstrated to have very good encryption performance, which is comparable with that of Salsa20/12 and Rabbit although the biggest gains in speed become evident after encryption of longer streams due to the cipher's costly key setup [72].

The cipher offers a considerable security margin although a vulnerability was found for key lengths longer than 226 bits. In 2008, Lee, Lee e Park proposed an linear mask cryptanalysis leading to a new vulnerability [73]. Two years later, Feng present a byte-guess attack [74].

5.2.4.2.4.5 Grain v1

The cipher was one of the simplest designs submitted to the project. The cipher aims at restricted hardware environments and it uses 80-bit keys and 64-bit initialization vectors. Its design combines an 80-bit linear feedback shift register (LFSR) and an 80-bit nonlinear feedback shift register (NLFSR).

Grain's design makes it ideal for hardware implementations since it allows execution of up to 16 encryption rounds in parallel. This property was, in fact the main reason the cipher was included in the eSTREAM portfolio because its security margin was estimated to be very tight due to a number of vulnerabilities discovered [65]. There are also side channels attacks presented on Chapter 7 of Strobel and Paar [75].

5.2.4.2.4.6 Mickey v2

The MICKEY (Mutual Irregular Clocking keystream generator) family of ciphers was designed for resource-constrained hardware platforms offering good performance, small implementations and high security [76], [77]. The version submitted to eSTREAM used 128-bit keys and its performance was reported to be bit lower than that of Trivium and Grain. The main reason the cipher was selected to the eSTREAM portfolio was its security because no negative cryptanalytic results against the cipher were found [65], [78].

5.2.4.2.4.7 Trivium

Trivium was the simplest design submitted to the eSTREAM hardware profile. The cipher has only 288 bit-long state, and it uses 80-bit keys and initialization vectors. Trivium can be implemented very efficiently in hardware (low gate count) but also gives very good results when implemented in software.

The cipher's simple design drew much attention in the cryptographic community and as of 2008 no attacks faster than the brute-force key search were known. However, the eSTREAM committee recommended that more time should pass before the cipher can be widely adopted in order to ensure that the cipher's simplicity is not its weak point [66], [65]. Indeed, since then, a number of attacks have been published on reduced-round versions of the cipher leaving its security margin to be rather slim. Indeed, since then, a number of attacks have been published on reduced-round versions of the cipher leaving its security margin to be rather slim. (i.e., see Priemuth-Schmid and Biryukov attack [79]). Daehyun Strobel and Paar also proposed an hardware side channel attack on Trivium on [75].

5.2.4.2.5 Suitability 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. For example, **Error! Reference source not found.** presents security requirements of the SSL protocol for different data bandwidth values [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.

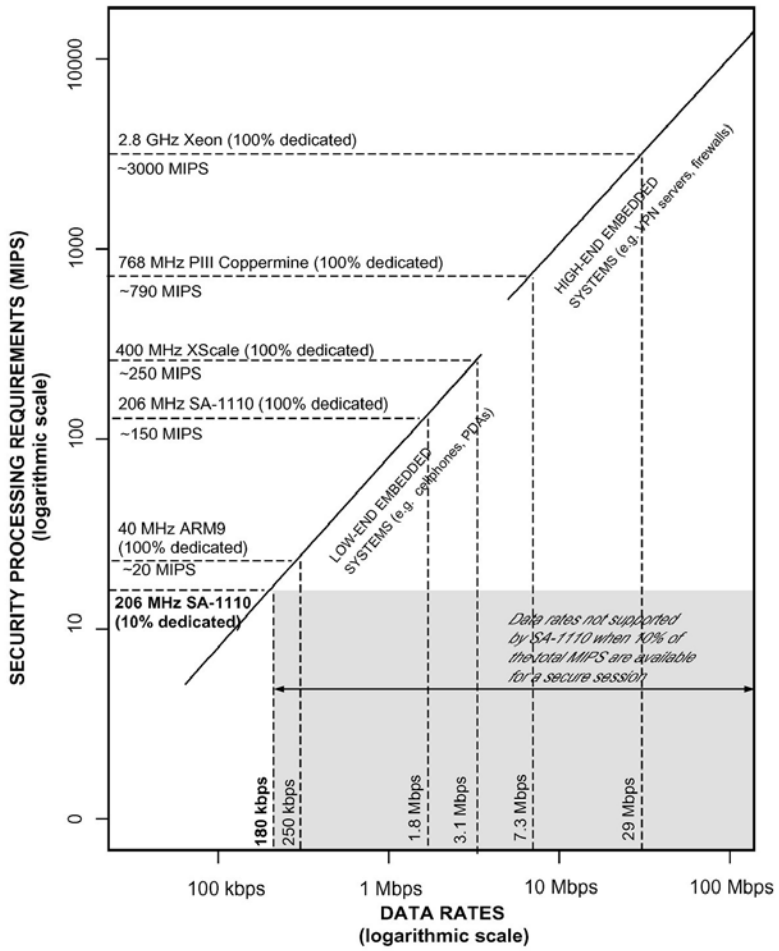


Figure 5-1: Processing requirements for the SSL protocol at different data rates [2]

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.

5.2.4.3 Message Authentication Codes

The ability to create a unique and non-forgeable digest of a message is of great practical importance. In particular, message authentication can be implemented by directly linking the sender's identity to the message's contents in form of a message authentication code (MAC). There are two general ways of implementing MACs: using cryptographic hash functions and running block ciphers running special modes such as, for example, cipher block chaining (CBC).

5.2.4.3.1 Cryptographic Hash Functions

There are two general applications of cryptographic hash functions: message integrity and message authenticity. In the former case, the function is used to produce a message digest (hash), which is a typically shorter representation of the message that is used to verify that the message's contents have not changed. In the latter case, a secret key is added to the function's input in order to relate the resulting digest, and hence the entire message, to the owner of the key.

Cryptographic hash functions, in order to be secure, are required to possess the following properties:

- Computing the hash is computationally easy.
- It is infeasible to recover the message for a given hash value.
- It is infeasible to find two different messages with the same hash.
- It is computationally hard to find two different messages for which the hash function produces the same digest (collision resistance).

Cryptographic hash functions can be used as ordinary hash functions since they have to be collision resistant and they have the compression property of producing small digests for large messages. This application is not common, however, because of typically higher computational complexity of cryptographic hash functions compared to the regular hash functions.

Message authentication with cryptographic hash functions can be performed by adding a secret key together with padding to the message and then by computing a hash of thus augmented message. HMAC is the most popular scheme of this sort and it is used in the IPsec and TLS protocols. The scheme provides both data integrity and authenticity, and it can work with a wide variety of cryptographic hash functions. HMAC is considered secure although an attack was recently devised that could forge HMAC authentication codes when used with the MD4 hash functions with only 2^{58} data complexity [81], [82], [83] and [84].

There has been a number of cryptographic hash functions developed and they differ in both their performance characteristics and in their security (understood as the ability to defend the core four properties outlined above). Similarly to ciphers, establishing the security of cryptographic hash functions is a lengthy process that requires years of analysis by international security experts. Thus, various official bodies such as the US National Security Agency, the European NESSIE project and the Japanese CRYPTREC project announced public calls for secure and efficient cryptographic hash functions. We outline the most important ones below.

- **SHA-1** (Secure Hashing Algorithm) is a cryptographic hash function designed by the NSA and standardized by NIST in the 1995. The function is a successor to SHA-0 which was identified with significant weaknesses. The function produces 160-bit digests in 80 rounds of computation and it remains widely used despite an improving range of attacks being discovered. Currently, NSA is phasing the function out in favour of its successor SHA-2. The function was also selected by the CRYPTREC project.
- **SHA-2** is a family of cryptographic hash functions (SHA-224, SHA-256, SHA-384, SHA-512) designed by NSA as a replacement for the SHA-1 function. The SHA-224 and SHA-256 functions operate 512-bit blocks of data and produce digests of 224 and 256 bits respectively, while the SHA-384 and SHA-512 functions operate on data blocks of 1024 bits and output digests of 384 and 512 bits respectively. Although, the SHA-2 function is extensively used in a range of applications and protocols including TLS, SSL, PGP and IPsec, NSA is already holding a competition for a successor to SHA-2. The functions were also selected by both the CRYPTREC and NESSIE projects.
- **RIPEMD-160** is an improved version of the RIPEMD function originally designed in 1996 as an improvement over the MD4 function. Although RIPEMD is a more general design that supports also digest sizes of 128, 256 and 320 bits, RIPEMD-160 is considered the most secure and was recommended for use by the CRYPTREC project.
- **MD2, MD4, MD5** is a family of cryptographic hash functions that were designed by Ronald Rivest of RSA Security in 1989, 1990, 1992 respectively. The functions produce 128-bit digests and they have been used extensively over the last decades combining good performance with security. However, due to a number of vulnerabilities discovered, MD4 and MD5 are not recommended for use anymore [85].
- **Whirlpool** is a cryptographic hash function whose design has many similarities to the Rijndael block cipher. The function produces digests of 512 bits and it was selected to the NESSIE portfolio of cryptographic primitives. Whirlpool originally had two earlier versions, Whirlpool-0 and Whirlpool-T that were identified with minor flaws. The name Whirlpool refers to the final, third, version of the function, which was also standardized by ISO (the ISO/IEC 10118-3:2004 standard). We are not aware of any realistic security flaws of the function.

Similarly to block ciphers, cryptographic hash functions are also characterized by their performance characteristics. Table 5-1 presents a comparison of data throughput and of the number of processor cycles needed per byte of input for all of the above presented functions. Readily, MD5 offers the highest data throughputs while Whirlpool is the slowest.

Function Name	MiB/Second	Cycles/Byte
MD5	343	5.8
SHA-1	131	15.3
SHA-256	132	15.1
SHA-512	118	17.0
Whirlpool	69	28.9
RIPEMD-128	208	9.6
RIPEMD-160	143	14.0
RIPEMD-256	220	9.1
RIPEMD-320	149	13.4
HMAC(SHA-1)	131	15.3

Table 5-1: Performance comparison of cryptographic hash functions (Crypto++ library benchmark)

5.2.4.3.2 Block Cipher Modes for Message Authentication

An alternative to custom-designed cryptographic hash functions is to use block ciphers in a special *mode of operation* that allows constructing a digest for the entire message. Modes of operation are techniques for encryption of messages with block ciphers in situations in which the message is longer than the cipher's block. Many different modes of operations exist. The most common ones are:

- **Electronic Code Book (ECB):** splits the message into sections equal in size to the cipher's block and encrypts it one by one. The use of the mode is discouraged due to its vulnerability to cut and paste attack.
- **Cipher Block Chaining (CBC):** splits the message in blocks and XORs each plaintext block with the cipher text block obtained in the previous round. A random initialization vector is used for the first round of the process.
- **Cipher Feedback (CFB):** the process starts by encrypting the initialization vector and XORing the first block of plaintext with the result of the encryption. In each subsequent round, the cipher text from the previous round is encrypted and then the corresponding plaintext block is XORed with the result.
- **Output Feedback (OFB):** In each round, the output from the encryption algorithm from the previous round is passed as the input for encryption in the next round and, at the same time, it is XORed with the current block of plaintext to form the current block of cipher text. The process is seeded with the initialization vector.
- **Counter Mode (CTR):** In each round, the sequence function (that can be a simple counter) is combined with the nonce / initialization vector (via XOR, added or concatenated) and with the key, the encryption algorithm transforms the current block of plain text into the current block of cipher tex. The process has similar characteristics to OFB, but also allows a random access property during decryption. CTR mode is well suited to operation on a multiprocessor machine where blocks can be encrypted in parallel. Furthermore, it does not suffer from the short-cycle problem that can affect OFB. The CTR mode (CTR) is also known as integer counter mode (ICM) or segmented integer counter (SIC) mode.

In addition to the encrypting modes of operation, there are also modes that result in computation of a message authentication code for the entire plaintext, as well as there are authenticating modes of encryption that combine both encryption with authenticity.

The most common MAC mode of operation is CBC-MAC. Similarly to CBC, the plaintext is split into blocks of equal size and then subsequent blocks are encrypted. The output from each step is XORed with the input to the subsequent step. The MAC of the message is the output of the last step of the process.

Several authenticated modes of encryption have been designed and six of them, namely OCB 2.0, Key Wrap, CCM, EAX, Encrypt-then-MAC and Galois Counter Mode (GCM), have been standardized in ISO/IEC 19772:2009. The most notable one is the GCM mode of operation since it has been proposed for use with AES (AES-GCM)[13]. GCM has more complicated structure than the basic modes presented above and it requires one block cipher operation and one 128-bit multiplication in the Galois field per each block (128 bit) of encrypted and authenticated data.

The main benefit of using block ciphers in MAC modes of operation is the ability to reuse the implementation of the block cipher to perform MAC calculation in addition to encryption. This fact is of great importance when memory constrained devices such as smart cards or small wireless sensor nodes are used. For example, the TinySec system uses the Skipjack cipher in CBC-MAC mode to implement message authentication [86].

As far as efficiency is concerned, authenticated modes of encryption offer lower speeds than the non-authenticated ones. However, it turns out that the use of block ciphers for MAC computation is less efficient than the use of dedicated cryptographic hash functions. This is evidenced by the comparison of the results presented in Table 5-1 and in Table 5-2.

Mode	MiB/Second	Cycles/Byte
AES/GCM (2K tables)	126	15.9
AES/EAX	73	27.5
AES/CBC (128-bit key)	131	15.3
AES/CBC (192-bit key)	110	18.3
AES/CBC (256-bit key)	97	20.6
AES/CTR (128-bit key)	165	12.2
AES/CFB (128-bit key)	131	15.3
AES/ECB (128-bit key)	132	15.2
AES/OFB (128-bit key)	124	16.2

Table 5-2: Performance comparison of chosen modes of operation of AES (Crypto++ library benchmark)

[13] In the RFC 5430, “Suit B Profile for Transport Layer Security (TLS)” the TLS version 1.2 requires that the two mandatory GCM suits (TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 and TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256) are preferred over the CBC (TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 and TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256) since the order in TLS ClientHello message is defined in the Compilance and Interoperability requirements section.

5.2.4.3.3 Suitability Discussion

Message authentication is an important building block of secure communication systems. It helps to ensure that contents of the message have not been tampered with and, by associating a shared secret with the MAC generation process, one can also implement origin authentication (data authentication). Similarly to block ciphers, however, choosing the right MAC technique for use in potentially resource-constrained environments requires consideration of performance issues. In particular, the following are the most important metrics that influence the applicability of a technique in the domain.

- Memory consumption - both ROM and RAM memory requirements have to be considered.
- Processing power - high-throughput systems require fast computation of MACs.
- Digest size - long messages require longer digest sizes in order to avoid collisions. On the other hand, attaching a long MAC to every radio packet in energy-constrained wireless networks might prove prohibitive.

The comparison of performance results presented in Table 5-1 and in Table 5-2 shows that calculating MACs with cryptographic hash functions can be faster than using AES. However, the use of 512-bit long digests makes it difficult for cryptographic hash functions to be applied to WSNs since the traffic patterns in these networks favour sporadic communications of short data packets. Therefore, taking into consideration the extra memory cost of implementing both the hash function and the block cipher, we recommend that either CBC-MAC is used for standalone MAC computation or that authenticated encryption scheme, such as GCM, be implemented. Alternatively, if the memory cost is not significant, the RIPEMD-160 function with 128-bit digests could be used.

5.2.4.4 Conclusions

Section 5.2 presented a range of security issues in networked embedded systems, typically lightweight devices, and discussed the cryptographic means affiliated. Two types of cryptographic approaches, i.e., asymmetric and symmetric, were discussed in the light of resource constrained architectures.

In terms of asymmetric cryptography, also known as public-key cryptography, ECC, RSA and EL-Gamal were discussed. ECC was commonly recommended by different standardization bodies mainly due to its small key size compared to other public key approaches such as RSA. However, as far as power consumption and speed are concerned, ECC does not provide a complete solution. For example, ECC is considerably faster in terms of digital signing as opposed to RSA where the contrary is true in terms of verification. Also, on a similar platform, ECC is slower than RSA during encryption while ECC outperforms RSA during decryption in terms of speed. Concerning El-Gamal, it does not seem to be a popular approach for cryptography in lightweight devices as, for example, it consumes much more energy during the encryption process as opposed to RSA. Eventually, a compromise has to be found depending on the type and frequency of secure operations in the system in addition to the processing and memory constraints. ECC will always have the advantage of low memory usage as it needs a small key size to provide a comparable security to RSA for instance.

A large number of symmetric ciphers have been designed to date and they vary in their security and performance characteristics. The security of a symmetric cipher cannot be easily established at design time and usually many years of exposure to public scrutiny are required in order to consider a cipher secure. On the other hand, performance characteristics can be measured and the best-performing cipher can be objectively selected. As a result, we conclude that AES (Rijndael) is the best candidate cipher for use in the pSHIELD middleware because of its extensive analysis by the world's best cryptographers and because of its excellent performance characteristics. The cipher is suitable for implementation on memory constrained devices and in hardware, 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).

We showed that public-key cryptography, although it offers a very flexible security model, cannot be universally applied in the domain of small embedded devices due to its resource requirements. Symmetric cryptography, on the contrary, is typically resource efficient and fast but it requires a secure channel for all communicating parties to establish a shared secret. As a result, hybrid approaches that combine both public-key and symmetric cryptography are recommended to be used instead.

5.2.5 Attacks

5.2.5.1 Pollard's rho Attack Versus Index Calculus

Between the difficulty of the forward and inverse operations at the centre of all popular asymmetric schemes is a big difference. In RSA, it is integer multiplication (forward) and factorization (inverse) that make the system work. In Diffie Hellman it's discrete exponentiation (forward) and log (inverse). In ECC it is point multiplication (forward) and the elliptic curve discrete logarithm problem (inverse).

In all of these cases, it is easy to see that the difficulty of the brute force approach to the inverse operation increases exponentially with the size of the key. Simply look at the number of values that must be tried; it doubles with each bit added to the key length.

ECC cryptosystems aren't vulnerable to index calculus attacks. Index calculus attacks rely on certain group properties not present in groups defined using elliptic curves.

Pollard's rho attack is a class of what are known as 'collision search' attacks. They also do better than brute force. But they also get a lot harder a lot faster than do the index calculus attacks, as the field size increases.

5.2.5.2 Known Attacks Against Elliptic Curve Cryptosystems

Here we present some known attacks against elliptic curve cryptosystems. The scope is limited to algorithms solving the elliptic curve discrete logarithm problem (ECDLP), i. e. to determine l given a point P and a point $Q = lP$, and it does not consider attacks against particular elements of digital signature algorithms based on elliptic curves.

1. **Naive Exhaustive Search**: The most simple approach to obtain l is to compute successive multiples of $P, 1P, 2P, 3P, 4P, \ldots$ until the result is equal to Q. In the worst case, this takes n steps, where n is the order of the point P.
2. **Baby-Step Giant-Step Algorithm**: This algorithm is a time-memory trade-off of the method of exhaustive search. It requires storage for about $\sqrt{n}$ points, and its running time is roughly $\sqrt{n}$ steps in the worst case.
3. **Pollard's Rho Algorithm**: This algorithm is a randomized version of the babystep giant-step algorithm. With some modifications it can be sped up to have an expected running time of $\frac{\sqrt{\pi n}}{2}$ steps and it requires only a negligible amount of storage.
4. **Parallelized Pollard's Rho Algorithm**: The original Pollard's rho algorithm can be parallelized so that when it is run in parallel on r processors, the expected running time is roughly $\frac{\sqrt{\pi n}}{2r}$.

5. **Pollard's Lambda Method**: Pollard also presented a lambda method for computing discrete logarithms which is applicable when l, the logarithm sought, is known to lie in a certain interval. In particular, when l is known to lie in a subinterval $[0, b]$ of $[0, n-1]$, where $b < 0.39n$, the parallelized version of Pollard's lambda method is faster than the parallelized Pollard's rho algorithm.
6. **Multiple Logarithms**: It turns out that if a single instance of the ECDLP is solved using (parallelized) Pollard's rho method, the following instances (for the same curve E and the same base point P) can be solved faster, since some of the necessary work has already been done in the previous steps. In fact, solving k instances of the ECDLP takes only $\sqrt{k}$ as much work as it does to solve one instance.

Hence, the best known attack against a single instance of the ECDLP is Pollard's rho algorithm and has an expected running time of $\frac{\sqrt{\pi n}}{2} \approx O(n^{1/2})$, which is fully exponential.

However, there are certain elliptic curves with special vulnerabilities that can be exploited by the following algorithms. These algorithms may have shorter running times as those mentioned before; therefore elliptic curves with these vulnerabilities should be avoided.

1. **Pohlig-Hellman Algorithm**: This algorithm exploits the factorization of n, the order of the point P, and reduces the problem of recovering l to the problem of recovering l modulo each of the prime factors of n. We can then recover l by using the Chinese Remainder Theorem. As a countermeasure, one should select an elliptic curve whose order is a prime or almost a prime (i. e. a large prime times a small integer).

2. **Supersingular Elliptic Curves**: In some cases, the ECDLP in an elliptic curve E defined over a finite field $\mathbb{F}_q$ can be reduced to the ordinary discrete logarithm problem (DLP) in the multiplicative group of some extension field $\mathbb{F}_{q^k}$ for $k \geq 1$. The DLP is the underlying computationally hard mathematical problem for the DSA and one can solve it using the number field sieve algorithm, which has a subexponential running time. To ensure that the reduction algorithm does not apply to a particular curve, one only needs to check that n does not divide $q^k - 1$ for all small k for which the DLP in $\mathbb{F}_{q^k}$ is tractable. In practice, it suffices to check this for $1 \leq k \leq 20$ when $n > 2^{160}$.

3. **Prime-Field Anomalous Curves**: If the number of points on a curve E over $\mathbb{F}_p$ is equal to p, the ECDLP can be solved efficiently. This attack can be avoided by verifying that the number of points on an elliptic curve is not equal to the cardinality of the underlying field.

4. **Curves Defined Over a Small Field**: For elliptic curves E with coefficients in $\mathbb{F}_{2^n}$, Pollard's rho algorithm for computing elliptic curve logarithms in $E(\mathbb{F}_{2^{nd}})$ can be further sped up by a factor of $\sqrt{d}$. For example, if E is a Koblitz curve, then Pollard's rho algorithm for computing elliptic curve logarithms in $E(\mathbb{F}_{2^m})$ can be sped up by a factor of $\sqrt{m}$.

5. **Curves Defined Over $\mathbb{F}_{2^m}$, m Composite**: The Weil descent might be used to solve the ECDLP for elliptic curves defined over $\mathbb{F}_{2^m}$ where m is composite. There exists some evidence that when m has a small divisor l, e.g. $l = 4$, the ECDLP can be solved faster than with Pollard's rho algorithm. Thus, elliptic curves over composite fields should not be used.

Let us take a look how secure elliptic curve cryptography is in practice. Certicom challenge [106] is one source that gives a notion about the security of ECC. The challenge is to compute the ECC private keys from a given list of ECC public keys and associated system parameters. The challenge consists of two levels:

- Level I: 109-bit and 131-bit challenge; considered to be feasible
- Level II: 163-bit, 191-bit, 239-bit and 359-bit challenge; expected to be computationally Infeasible

Of the Level I challenges, the 109-bit ECC2K-108 challenge has been solved in April 2000 and the 109-bit ECCp-109 challenge has been solved in November 2002. All other challenges have been unsolved until March 2003. The computational cost to solve these challenges met the expected values according to Certicom

Concrete recommendations based on the expected cost of the Certicom challenge have been presented by Lenstra and Verheul in [103]. They predict which elliptic curve key sizes can be considered as secure until which year using a model that incorporates technological and cryptanalytical advances. Table 3 presents their recommendations for future years.

Year	Elliptic Curve Key Size	Infeasible Number of Mips Years	Corresponding Number of Years on 450MHz Pentium II PC
2002	139	$2,06*10^{10}$	$4,59*10^{7}$
2003	140	$3,51*10^{10}$	$7,80*10^{7}$
2004	143	$5,98*10^{10}$	$1,33*10^{8}$
2005	147	$1,02*10^{11}$	$2,26*10^{8}$
2006	148	$1,73*10^{11}$	$3,84*10^{8}$
2007	152	$2,94*10^{11}$	$6,54*10^{8}$
2008	155	$5,01*10^{11}$	$1,11*10^{9}$
2009	157	$8,52*10^{11}$	$1,89*10^{9}$
2010	160	$1,45*10^{12}$	$3,22*10^{9}$
2011	163	$2,47*10^{12}$	$5,48*10^{9}$
2012	165	$4,19*10^{12}$	$9,32*10^{9}$
2013	168	$7,14*10^{12}$	$1,59*10^{10}$
2014	172	$1,21*10^{13}$	$2,70*10^{10}$
2015	173	$2,07*10^{13}$	$4,59*10^{10}$
2016	177	$3,51*10^{13}$	$7,81*10^{10}$
2017	180	$5,98*10^{13}$	$1,33*10^{11}$
2018	181	$1,02*10^{14}$	$2,26*10^{11}$
2019	185	$1,73*10^{14}$	$3,85*10^{11}$
2020	188	$2,94*10^{14}$	$6,54*10^{11}$
2021	190	$5,01*10^{14}$	$1,11*10^{12}$
2022	193	$8,52*10^{14}$	$1,89*10^{12}$

Table 3: Minimum key size for elliptic curve cryptosystems providing a sufficient level of security

5.2.6 The Controlled Randomness Protocol

5.2.6.1 Introduction

In real world applications of cryptographic protocols, the key management problem refers to the life cycle management of cryptographic keys. It includes the necessary operations for key generation; distribution; storage; replacement and exchange; usage; and destruction. In order to retain specific security level, keys used in cryptographic algorithms and protocols must be periodically refreshed i.e., new keys are exchanged between communicating parties and old keys are replaced. These precautions ensure that only a specific amount of information is encrypted under the same key and thus, the exposure of information is minimized in case a key is leaked.

Key agreement is the process by which two or more parties agree on a common cryptographic key for a specific timeframe. Key transport is the process by which the agreed key is transferred to the participants. In many scenarios, the two processes occur simultaneously: the participants exchange information by which they both set and exchange the key(s) to be used (or some parts of it). In many scenarios, the key agreement and transport occur as exchange of control messages through a control channel. This channel does not interfere with the data channel in where actual secure data exchange takes place. A public-key cryptosystem (PKC) is commonly used in such setups in order to securely exchange through the control channel the symmetric-key cryptosystem (SKC) encryption/decryption keys used to securely exchange data within the data channel. The latter keys are often called ephemeral or session keys, since their lifetime spans a specific time period i.e., a session and then they are disposed.

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

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

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

5.2.6.2 Protocol Description

Conventional cryptographic schemes operate under the assumption that at most one key is active in any time moment. There is only one exception to this assumption. This is the transition periods when changing a cryptographic key. In these cases, at most two keys can be active in order to cope with delayed messages. We propose a novel approach of having more than one key at any given time moment. The approach is based on the concept of “controlled randomness” i.e., randomly using keys in a controlled environment. The concept of “controlled randomness” can be utilized in any protocol that uses temporal (ephemeral) keys. It increases protocol security with minimal computational overhead.

5.2.6.2.1 Protocol Definition

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

The Controlled Randomness Protocol works as follows. Within the time period t every cryptographic key k1, k_2... k_n is valid and can be used. The sender chooses with a uniform distribution a random

integer i and encrypts the input data using the key k_i. The receiver has access to a secret mechanism and upon receiving a ciphertext c_i can deduce which of the possible keys was used for the encryption and thus, use the correct one to decrypt the ciphertext. The CRP does not dictate how all these keys are transferred to the receiver. It can be through a control channel using a PKC scheme, or an SKC with master key, or any other method. The CRP dictates how all these keys are used and reused within a time frame composed of many conventional sessions.

Two different methods are proposed for deriving the index, j, of the secret key used for a given ciphertext. The first method is using a synchronized random number generator (RNG) in both the sender and the receiver for the indexes.

The second method involves usage of a Keyed Hash Function (KHF) also known as Message Authentication Code (MAC). The sender and the receiver agree on a set of n encryption keys for a chosen encryption algorithm as usual and additionally on a set of n keys for computing MAC. The sender further uses an RNG. In this cases, the sender works as follows for every plaintext m:

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

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

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

5.2.6.2.2 Advantages of CRP

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

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

The CRP allows in all above scenarios to extend the lifetime of each key way beyond the time of a conventional session. Further, it allows less frequent exchanges of messages in the control channel (if one is implemented), since less keys are needed to achieve a specific security level for a specific timeframe. An attack on the classical key management protocol with a master key of n bits has complexity $O(2^{2n/3})$; an attack on the RNG for the controlled randomness protocol with l keys has complexity $O(l2^m)$ (usually for m, the period of RNG, it holds m >> n); and an attack on the KHF method has a total complexity of $O(l(2^p + l2^{n/2}))$ where p the size in bits of the MAC keys.

5.2.6.2.3 Problem statement

It is argued that the KHF (MAC) method leads to an efficient implementation in the case of combining a symmetric encryption algorithm with KHF operations. In the simplest scenario, a 0,05l overhead is introduced on average for superior security. In a more complex scenario, this overhead can be lowered to 1%. This is achieved by performing the key detection function every few packets instead of each packet, as in the simpler scenario. We validate this argument for overheads in the SUN SPOT platform that is going to be use as the micro node in the pShield.

We denote c the cost of encryption/decryption and m the cost of the keyed hash computation. Suppose we have l valid key pairs. A pair contains two keys: one key for the encryption and one key for the hashing.

In the simplest scenario of adding one hash at every encrypted packet, the total cost t becomes t = c + lm/2 = c(1 + l/2). If we compute a MAC every k packets, then the total average cost becomes t = ((k - 1)c + (c + lc/2))/k = kc/k + cl/(2k) = c(1 + l/(2k)). Turning k = 10l reduces the overhead factor to 0.05 i.e., only 5%.

We are free to decide the specific *l* and *k* for our setup in such a way that offers sufficient security without sacrificing performance. As an arithmetic example, in case of small packets of 1 KB and *l* = 4, one can decide to alternate keys every *k* = 20 packets i.e., every 20 KB of input for a theoretical overhead computation of 10%. Recall that before mounting a cryptanalytic attack, an eavesdropper must first identify which block(s) are encrypted under which key; all she can now derive are larger blocks of data (20 KB instead of 1 KB) that are encrypted under a same key. However, and most importantly for security, the same *l* = 4 keys can be used without worrying of the security for significantly larger periods of time compared to the sum of time of serial use of each key. The attacker will be struggling to identify which of the four keys might have been used for each block of the 20 KB blocks of data in order to classify them as encrypted under the same key as to then mount a cryptanalytic attack.

5.2.6.3 Implementation

We implemented the CRP on the SUN SPOT devices in order to validate the theoretical performance figures. We also implemented the traditional approach of serial usage of encryption keys. We used the existing cryptographic algorithms and hashing functions provided by the platforms SDK.

We performed tests on three CRP variants. The difference between them is the underlying keyed hash algorithm used: HMAC/SHA1, HMAC/SHA256, and AES-CMAC. We tested the performance of these variants in both platforms for *l* = 4, 8, 16 and *k* = 1, 4, 8, 16. In these scenarios, *k* is the length of the key change window i.e., the number of packets (segments) encrypted under the same key before the sender switches to a new random key from the existing key pool. We analyze the results in the following paragraphs.

5.2.6.4 Performance Analysis

Error! Reference source not found. presents the performance of CRP on the SUN SPOT device. The numbers denote the average time for the decryption of a segment with size of 16 KB. We used 500 random generated packets for every test and we calculated the mean time for the packet decryption.

5.2.7 Optimization

Security in embedded applications has become a challenge. During the last years, all efforts have been focused on developing encryption/decryption algorithms capable of running on resource constrained systems.

One of the most critical scenarios is where wireless sensor networks (WSN) are distributed. The proliferation of this kind of networks has prompted increased interest in secure communications for embedded devices. Processor speed, bandwidth, energy usage, ROM and RAM size are the most relevant constrained resources to take into account before encrypting data.

The Advance Encryption Standard [87] (AES) is a strong encryption method, with reasonable performance and size. Nowadays, it is the standard used by the U.S. government to protect sensitive information because of its security, ease of implementation and low memory requirements. It is the successor of Data Encryption Standard (DES) which couldn't be considered as safe because of its short key length (only 56 bits). The inclusion of AES in the IEEE 802.15.4 [88] standard makes this encryption method ideal for use in WSNs.

5.2.7.1 Advance Encryption Standard – Basic concept [89]

Rijndael Cipher, developed by Joan Daemen and Vincent Rijmen was accepted as the Advance Encryption Standard on November 26.2001. It specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt (encipher) and decrypt (decipher) information.

AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits. Considering the resource constrains, only a key length of 128 bits will be discussed.

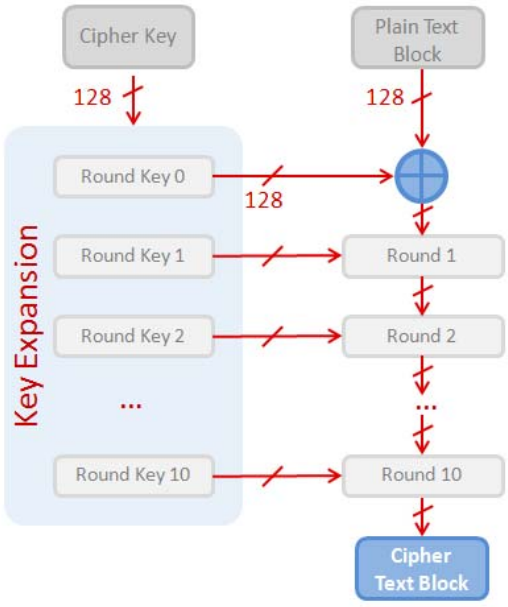


Figure 5-2: AES Algorithm Structure

Figure 5-2 describes the structure of AES algorithm. It consists of ten rounds of encryption where the 128-bit key is expanded into eleven so-called round keys. Each round includes a transformation using the corresponding cipher key to ensure the security of the encryption.

A series of permutations and substitutions are applied to encrypt the plain text and decrypt the cipher text (see Figure 5-3).

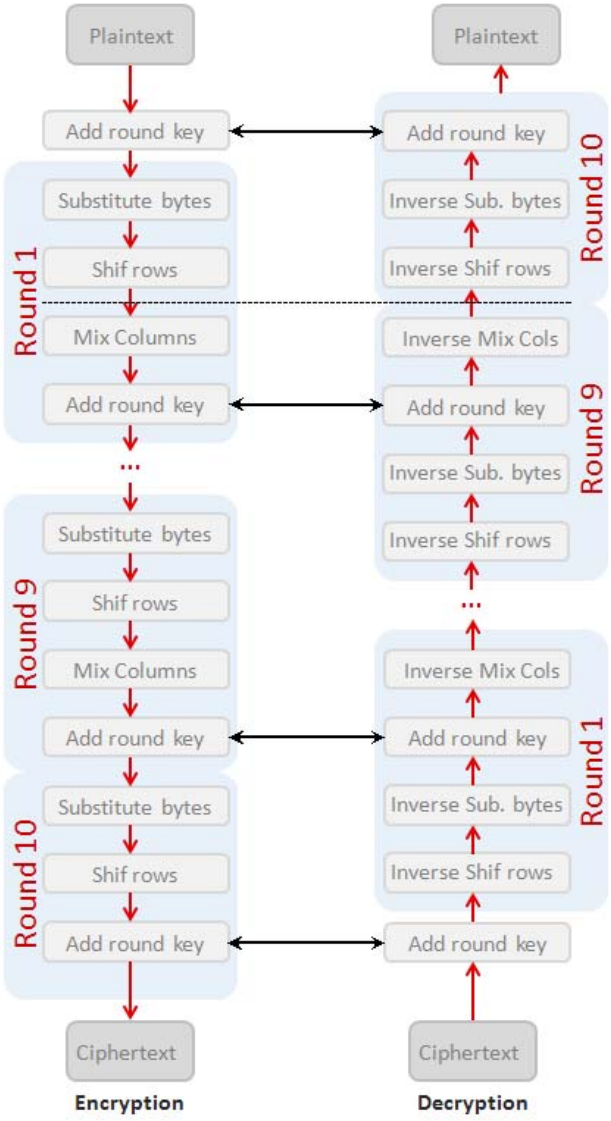


Figure 5-3: Overall structure of AES [90] – Encryption/Decryption process

Both the key and the input data (called “state”) are structured in a 4x4 byte matrix. There are four main transformations used in this process:

- ***Substitute bytes***: this transformation is a non-linear byte substitution which operates independently on each byte of the State using an invertible substitution table (S-box).

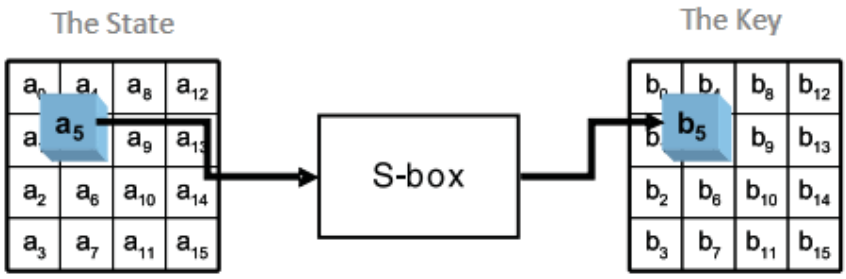


Figure 5-4: AES - "Substitute bytes" operation

- ***Shift rows***: the bytes in each of the 4 rows in the state are rotated by (n - 1) where n represents the row number from 1 to 4.

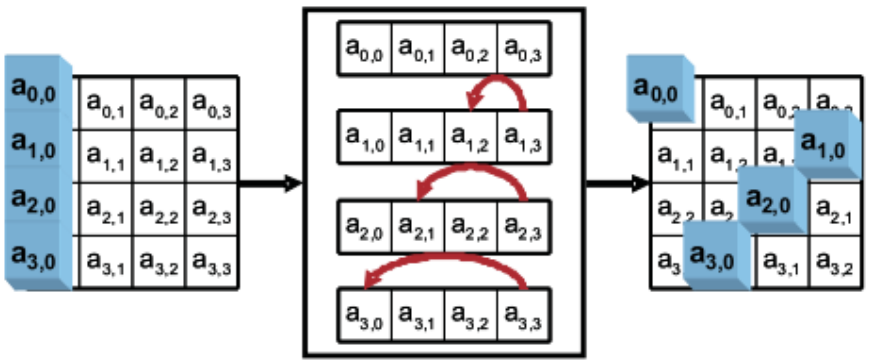


Figure 5-5: AES - "Shift rows" operation

- ***Mix columns***: Opposed to the "Shift rows" operation, which works on rows in the 4x4 "state" matrix, the "Mix columns" operation processes columns. To summarize this operation and reduce its complexity, it is important to know that this transformation combines each column of the state matrix using an invertible linear transformation.

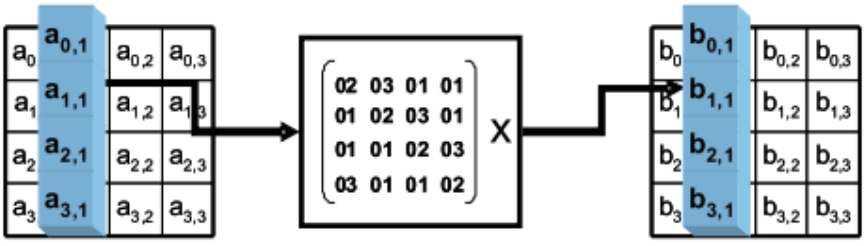


Figure 5-6: AES – "Mix columns" operation

- ***Add round key***: in this transformation, the round key is added to the state through a bitwise exclusive-or operation.

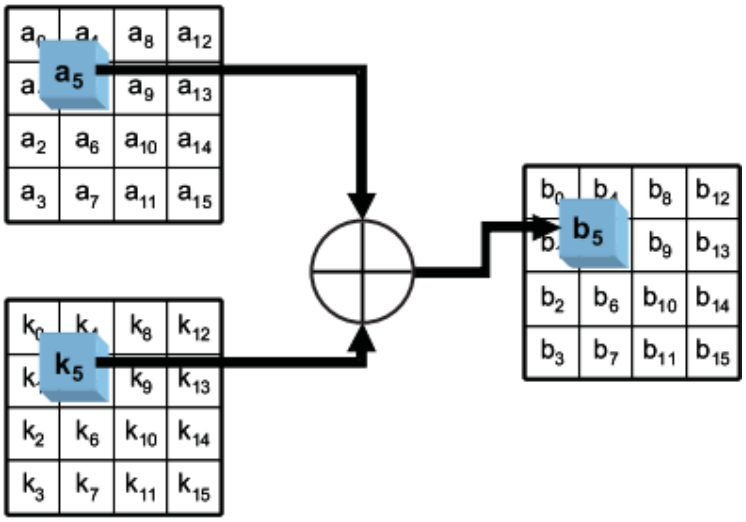


Figure 5-7: AES – "Add Round Key" operation

The cipher key is expanded to generate a different key for each round so during this process, the 128 bits of the original key are expanded into eleven 128-bit round keys. Two transformations are needed to achieve the expansion:

- ***SubWord***: this transformation is similar to SubBytes operation, where each byte in the word is substituted with a byte from a substitution the table (S-box).

- ***RotWord***: this transformation cyclically shifts the bytes of a word one position left.

The "*KeyExpansion*" routine uses an array "*Rcon[]*", called the round constant table. These constants are 4 bytes each to match with a row of the key schedule table.

5.2.7.2 Optimizations

As it has been commented previously, AES is an open standard that combines an extremely high level of security with computational efficiency. The algorithm consists of Exclusive-OR functions combined with matrix operations and is a mathematically 'clean' design which avoids the risk of 'back doors' to unauthorized users.

All embedded systems have limited resources in terms of memory size, processor performance and power consumption. Thus, hardware and software optimizations are needed to protect data transmission over insecure channels without compromising the system resources.

Because of its efficiency, this algorithm is appropriate for both hardware and software implementations. A hardware design of AES offers a superior performance with higher throughput but, currently, most AES algorithms are implemented in software, despite that the secret key is more vulnerable to attacks and some internet applications don't accept the maximum speed achievable by this implementation.

5.2.7.2.1 Overview

Software implementations can be performed through programming languages such as C, C++, Java and assembly language and they are usually implemented on general purpose microprocessors, signal processors and smart cards.

Hardware implementations ensure not only greater physical security but also a higher speed compared to software implementations. Thus, it is useful in those scenarios where a wireless security is needed like in military communications or mobile telephony, where one of the most challenges is the speed of communication. Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGA) are the most common architectures where AES implementation can be performed through hardware descriptions languages such as VHDL or Verilog HDL.

Some parameters like the minimum area, minimum power consumption or maximum throughput can determine the optimum hardware architecture for AES implementation. There are hardware environments where the maximum area of a cryptographic unit is limited by the available fabrication technology, power consumption, cost (the area of an integrated circuit affects directly to its cost) or any combination of these factors.

Support for feedback modes of operations, such as CBC and CFB, or non-feedback modes of operations, such as ECB and CTR modes, also determines the hardware architecture. Non-feedback modes can encrypt all blocks in parallel, whereas in feedback mode, all blocks must be encrypted sequentially.

The optimization also depends on whether the system must support only AES encryption (e.g., in the block cipher modes of operation that only require encryption, such as CTR and CFB modes) or both, encryption and decryption (e.g., in the modes that require both operations, such as ECB and CBC).

The semiconductor technology also influences on the optimization. ASICs are faster than FPGAs because of the delays introduced by the circuitry required for reconfiguration but the design cycle in PFGAs is significantly shorter than in ASICs since the physical design, fabrication and testing phases (for physical defects) are not needed.

Other factor to be considered during the optimization is the security level that must be implemented, that is, the resistance to side channel attacks, such as Deferential Power Analysis, Timing Analysis, etc.

5.2.7.2.2 Hardware optimizations

Several architectures can be adopted to implement the encryption/decryption unit. The major factors defined in 5.2.7.2.1 determine the best hardware solution for AES implementation:

- ***Basic iterative*** architecture only can encrypt one block of data at a time. The number of clock cycles to encrypt a single block of data is equal to the number of cipher rounds.

- ***Loop unrolling*** architecture reduces the number of clock cycles necessary to encrypt a single block by a factor of K because the combinational part of the circuit implements K rounds of the cipher, instead of a single round. Thus, the encryption throughput improves slightly and the latency is reduced, although this entails that the circuit area increases almost proportionally to the number of unrolled rounds.

- ***Pipelining*** architecture can process several blocks of data at the same time, that is, it can parallel processing multiple streams of data. This architecture improves throughput but also increases the area of the circuit, although this increase could be minimum if the most suitable hardware architecture is implemented for the selected cipher mode (feedback, non-feedback)

- ***Shared resources*** can significantly reduce the circuit area by using the same functional unit to process two or more parts of the data block in different clock cycles.

Different directions have been followed in order to improve and optimize cryptographic technologies into hardware systems.

- Development of high-speed architectures to improve the throughput and reduce the latency.
- Development of compact architectures to optimize the minimum area. This effort led to the emergence of architectures with 64, 32, and even 8 bit data paths [91], [92], [93]
- Optimization of basic operations of AES, including logic only implementation of SubBytes, and optimizations and decompositions of the MixColumns and InvMixColumns transformations.
- Development of different architectures for the entire encryption/decryption unit [94].

Hardware architectures can be optimized in terms of speed (data throughput) or Silicon area. The higher data throughput can be achieved with a FPGA or an ASIC platform since they can provide hardware acceleration.

Different studies and research had been carried out in order to compare and analyze the performance and main features of implementations of cryptographic transformations in ASICs, FPGAs and microprocessors.

Table 5-4: Features of implementations of cryptographic transformations in ASICs, FPGAs and microprocessors [95]

	ASICs	FPGAs	Microprocessors
	Performance Characteristics		
Parallel processing	yes	yes	limited
Pipelining	yes	yes	limited
Word size	variable	variable	fixed
Speed	very fast	fast	moderately fast
	Functionality		
Algorithm agility	no	yes	yes
Tamper resistance	strong	limited	weak
Access control to keys	strong	limited	weak
	Development process		
Description languages	VHDL, Verilog HDL	VHDL, Verilog HDL	C, C++, Java, assembly language
Design cycle	long	moderately long	short
Design tools	very expensive	moderately expensive	inexpensive
Maintenance and upgrades	Expensive	Inexpensive	inexpensive

5.2.7.2.3 Software optimizations for embedded devices and wireless sensor networks

In this section, several alternatives of implementing AES algorithm will be described. Some of them will be implemented in a wireless platform to find out if there are significant differences between the original algorithm and the one with some of the optimizations implemented.

During last years, several works have been carried out taking into account the increased use of mobile applications and the use of sensitive data. Embedded devices have very limited resources (CPU time, memory, battery power...) so it is essential implement security protocols in an efficient way.

The IEEE 802.15.4 standard was first released in 2003 and revised in 2006. It includes Wireless Medium Access Control as well as Physical layer specifications. Since most of the WSNs fall within the category of LR-WPANs, compliance with this standard ensures reliability, compatibility and scalability of the network. There are a total of 8 security modes and four of them offer data confidentiality through AES encryption cipher.

IEEE 802.11i security standard suite [97] has been developed as a replacement of the highly vulnerable Wired Equivalent Privacy (WEP) to provide the 'best' security for 802.11 wireless local area network (WLAN). IEEE 802.11i standard combines Counter (CTR) mode privacy and Cipher Block Chaining Message Authentication Code (CBC-MAC) and has been designed as a long term security solution. The CCMP protocol is based on Advance Encryption Standard (AES) cipher and it offers robust encryption and message integrity as proved in [98] and [99]. CCMP, in the context of Wi-Fi security, is often referred to as AES-CCMP or simply AES

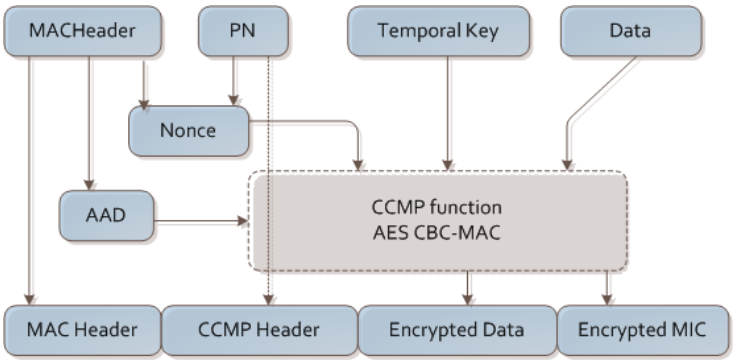


Figure 5-8: CCMP encapsulation process

Although CCMP is an efficient algorithm, it requires a powerful hardware to support the operations. Some optimizations already implemented by Gladman [96] have been studied in order to implement some of them to check the performance of the system, mainly in terms of power consumption:

- ***Code adaptation***: is useful to reduce code size and to improve the performance of the system. The original code, that supports key length of 128, 192 and 256 bits, can be adapted to AES-128, since the resources available in the platform are reduced and no other key length is considered.

- ***Varying data type size***: is possible with 32-bit or 64-bit processors. The effects of this optimization are more visible in "AddRoundKey" transformation, where the exclusive-or operations are faster since the registers are longer than 8-bits.
  The AddRoundKey transformation is very used so the effects on the performance should be clearly visible.

- ***Function-inlining***: consists of reorganizing the code to implement it as a single function instead of as discrete functions. Thus, the system doesn't have to save the state of the function onto the stack and the retrieve it.

  Although this optimization is very common, the code size might increase substantially during the implementation if some aspects, like repeated code segments or loop structures, are no taken into account.

- ***Loop Unrolling***: consist of unrolling the loops. Thus, instead of using a loop to iterate multiples times and an index to perform the same operation on different sets of data, the code is modified to eliminate all the rounds simply copying multiple times the same code to avoid calculating the array index. Although it is a common SW application, it can be rarely implemented due to the code size is substantially incremented.

- ***Reducing Memory Moves***: is possible if the code is restructured to reduce data movement. Thus, the operations, to copy data from one memory location to another, can be reduced simply saving the output of one transformation function in a location used as input for the next transformation.

- ***Using Pointers and eliminating Local Buffers*** in functions to reduce memory usage. Thus, it is possible to improve the efficiency since there are less copy-data operations.

- ***Using Global Variables*** since their address can be pre-computed before runtime by the compiler.

- ***On-the-fly-key generation*** is a technique useful when the performance is less important than the memory usage. This solution proposes generating the key "on the fly" in each round.

- ***MixColumns with 16-bit Memory Writes*** consists of modifying the MixColumns transformation to reduce the number of memory writes. The implementation will compute a 16-bit entry for the state by using 8-bit shift and an OR operation on two sets of four 8-bit values. This optimization is only effectible if the cost related to the speed in terms of memory writes is higher than the cost of the bitwise-or and bitwise-shift operations.

  Although CCMP is an efficient algorithm, it requires a powerful hardware to support the operations.

To perform the test, a proprietary wireless platform has been needed. It is a simple device with a microcontroller, a transceiver and an antenna to send/receive the information to/from an Access Point.

Table 5-5: Wireless platform – Microcontroller features

Microcontroller		
Supply voltage range		1.8V to 3.6V
Power Consumption	Active Mode	330uA (2.2V@1MHz)
	Standby Mode	1.1uA
	Off Mode	0.2uA
16-bit RISC Architecture		
Frequency		8MHz
Flash		48KB
SRAM		10240B

Table 5-6: Wireless platform – Transceiver features

Transceiver		
Frequency Band		387 – 464 MHz
16-bit RISC Architecture		
Frequency		8MHz
Flash		48KB
SRAM		10240B
Sensitivity		-116 dBm at 0.6kBaud
Current Consumption	TX	34mA@3.6V
	RX	17mA@3.6V
	Sleep mode	200nA

Although AES is one of the most suitable cipher solutions to be implemented in ES, the proprietary platform has limited resources so some SW optimizations were necessary to transmit protected data.

Figure 5-9: Proprietary wireless platform – ISM Band 433MHz

The original AES algorithm is too heavy so it couldn't be implemented because of memory constrains. The first changes applied were focused in reducing stack usage. All functions were revised to substitute the local buffers by pointers. Also, some variables were declared as global because they were accessed by multiple functions at all stages of the encryption/decryption. After these changes, the code could be compiled and all data transmitted was protected by AES algorithm.

Once the problem with the stack was solved, other optimizations were implemented to improved the performance of the system and reduce the power consumption. The code was adapted to AES-128 and some functions were modified to reduce the memory movements, always taking into account that the wireless platform is memory limited, so the proposed changes couldn't be focused on improving performance, compromising the memory resources.

This test aimed to check if AES algorithm was suitable to be embedded in a platform with resource constrains. Although some optimizations were needed, finally data could be encrypted/decrypted without comprising the power consumption.

5.2.8 Framework

5.2.8.1 Technologies Selection

5.2.8.1.1 Environment

In this section we present the TinyOS and Linux based environment used to develop and test the cryptographic technologies used in the low cost and power nodes.

The TinyOS is a free and open source operating system (OS) and platform that is designed for WSNs. It is an embedded operating system, written in the nesC Programing language as a set of cooperating tasks and processes. NesC is actually a dialect of the C programming language that is optimized for the memory limitation of the sensor networks [101], [102].

In the following diagram (Figure 5-10) the TinyOS tool chain is presented.

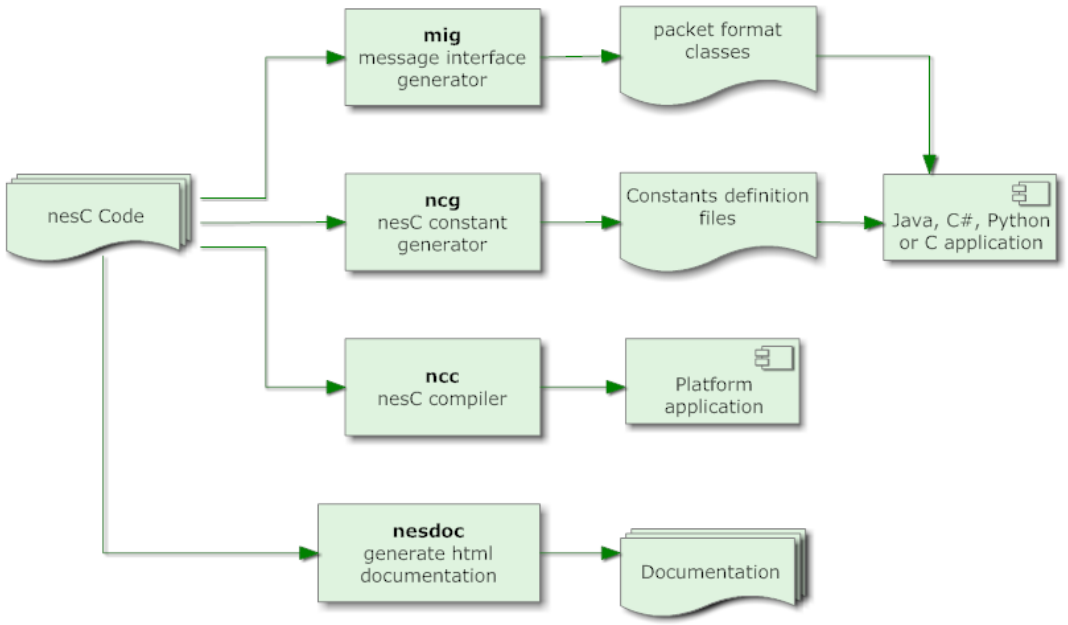


Figure 5-10: TinyOS tool chain diagram

In the development of TinyOS application, the NesC code is translated into a single C program, cross-compiled and transferred to the selected WSN Platform. In addition, an application written in C, Java, C# .Net or Python, than can run on Linux and Windows operating system, will communicate with the WSN nodes. This application needs to be using the same packet format and constants as the TinyOS application.

With the use of the message interface generator (**mig**) and the nesC constant generator (**ncg**), the information needed for communication is synchronized between the low power node running TinyOS and the power node, with the Java application running in a Linux distribution.

5.2.8.1.2 Methodology

For the execution of the cryptographic evaluation, we configured the environment according to the following scenarios:

- Single Low power node cryptographic evaluation;
- Single Power node cryptographic evaluation.

The configuration for each scenario is presented in detail in the next sections.

5.2.8.1.2.1 Single Low power node cryptographic evaluation

In this scenario, the objective is to measure the performance of the cryptographic operations inside the Telosb node, running TinyOS 2.1.1. To remove the overhead of radio communication (with the associated uncontrolled factors like media access), all the communication is made from the low power note to the laptop via USB. The aggressive low power mode of TinyOS was disabled during the evaluation for a direct relation between the operation time and the energy consumed.

In the next figure, the Low power test environment is presented and described.

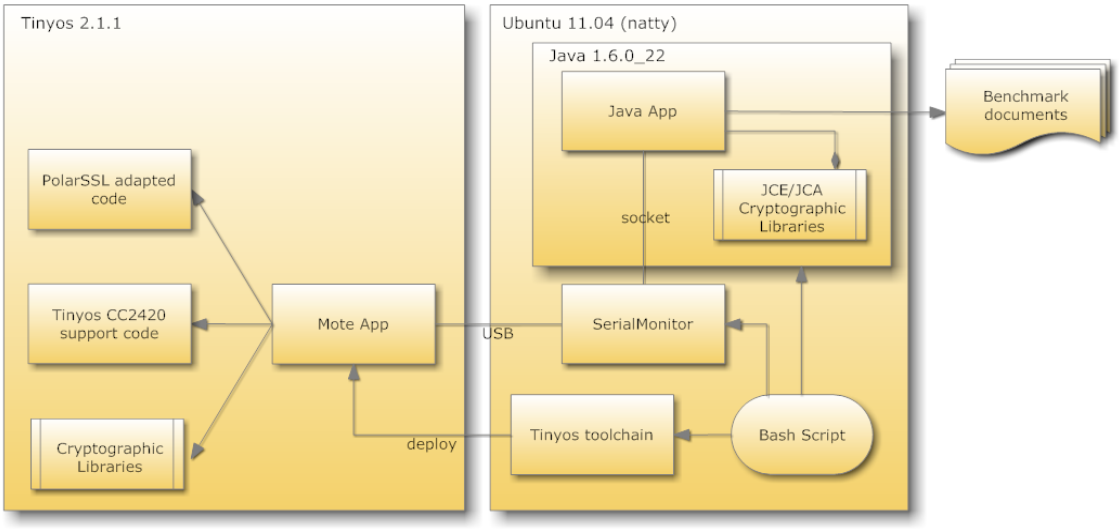


Figure 5-11: Low power node test environment

A Bash script compiles the Java and the nesC applications and transfer to the first mote attached to the Linux system. Then the SerialMonitor and the Java application start. The several different messages transmitted by the Telosb mote using USB port contained the private and public key, signatures and signature verification results. The Java program processes all the messages, calculates the time of operation, verify the results and logs the information to an excel “friendly” file (CSV), in tables in dokuwiki syntax, to a text file and to the console (all these elements are presented in the previous diagram as benchmark documents). For example, the system executed 30 rounds by default for each ECC domain curve, taking 9 to 15 minutes each.

All CSV files were imported to excel and the statistic metrics analysed. The standard derivation was small inside each batch and between test days.

The USB was used to prevent bias from the radio use.

All CSV files were imported to excel and the statistic metrics analysed. The most relevant results are presented in section 5.2.8.1.4.

5.2.8.1.2.2 Single Power node cryptographic evaluation

The development of benchmark applications were made, with similar excel and wiki “friendly” output to the previous scenario. The testing revealed some non-compliance with standards and even some implementation bugs that forced that the working version of some libraries changed to the last developer trunk. A Unix bash script ran all the tests under Ubuntu 11.04 32 bits during the night. The daily batch results during a month, running under different conditions (single mode, multi user mode; different kernels) were analysed.

The next figure illustrates the Power node environment.

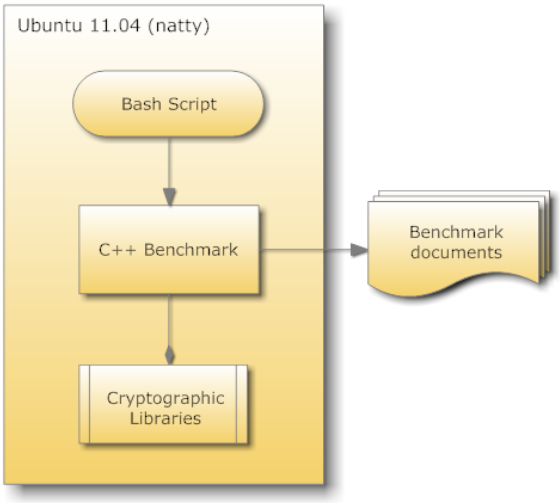


Figure 5-12: Power node environment

The base hardware was a Dell Vostro laptop that ran without battery and with power management disabled. In the next section, we describe the cryptographic libraries used for the pSHIELD evaluation.

5.2.8.1.3 Cryptographic Libraries

In this section, we present a non-exhaustive 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 useful for the pSHIELD project. Our focus was nesC, C, C++ and Java implementations.

In the next diagram, the libraries and their dependencies are presented.

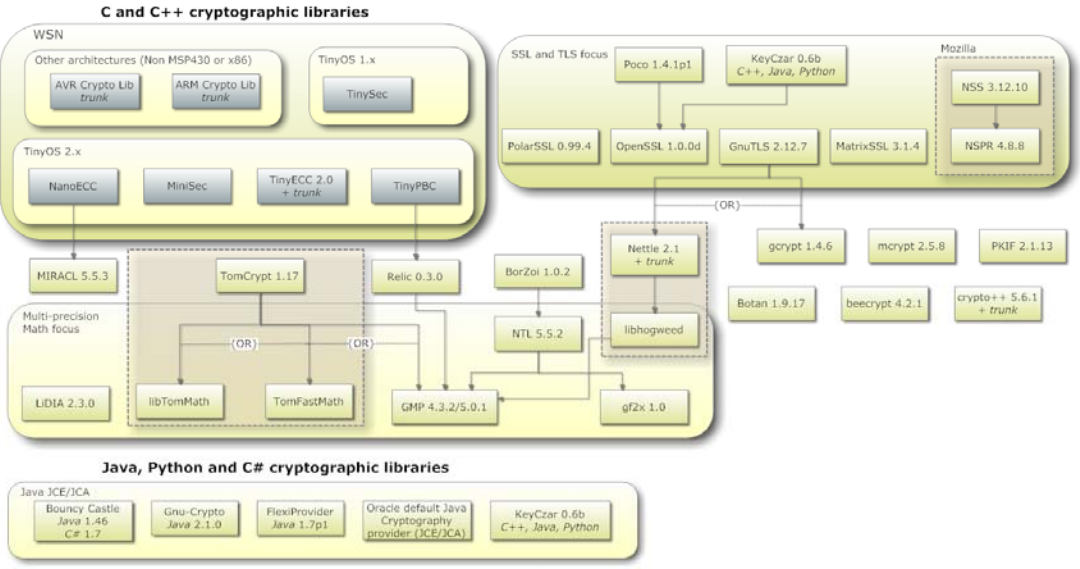


Figure 5-13: Open-source cryptographic Libraries with dependencies

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 cryptographic library from the previous diagram, in detail in the next sections.

5.2.8.1.3.1 AVR-Crypto-Lib

The AVR-Crypto-Lib is a library providing implementations of cryptographic algorithms for the AVR 8-bit micro-controller family. Due to the special limitations of micro-controllers, reference or "normal" optimized implementations are not usable. Therefore the developer team provides special implementations which respect the extreme limited resources of micro-controller applications.

Currently AVR-Crypto-Lib provides block cipher, stream cipher, hash functions, MAC functions and Pseudo Random Number Generators (PRNGs). Current work includes adding hash functions from

SHA3-competition for new good hash functions and working on defense against implementation attacks (such as DPA) and on implementing asymmetric crypto primitives (especially elliptic curve cryptography).

The AVR-Crypto-Lib is licensed under “GPLv3 or later”.

5.2.8.1.3.2 ARM-Crypto-Lib

The ARM-Crypto-Lib is a library providing implementations of cryptographic algorithms for the ARM micro-controller family. The ARM and AVR Crypto Lib are related, with common developers and hosted in the same domain.

5.2.8.1.3.3 Basicrypt

Basicrypt benchmark package contains standard and elliptic curve code for Diffie-Hellman key exchange, digital signature algorithm, EL-Gamal and RSA encryption/decryption. Standard algorithms can be used with various key lengths (1024, 2048, and 3072), while for elliptic curve variants parameter files are defined according to fields and curves recommended in NIST DSS standard FIPS 186-2. Input text files (input_small.asc and test.asc) that we used when running the benchmarks (where it was applicable) are taken from MiBench benchmark suite.

Public-key benchmarks in Basicrypt package were written using MIRACL C procedures for big integer arithmetic. The MIRACL library consists of over 100 routines that cover all aspects of multiprecision arithmetic and offer procedures for finite field elliptic curve operations.

5.2.8.1.3.4 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.

BeeCrypt support RSA, DSA, DHAES, AES, Blowfish, MD5, SHA-1, SHA-256, SHA-384 and SHA-512. There are no patent or royalty issues associated with BeeCrypt.

5.2.8.1.3.5 Botan

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

The Botan public key suport include RSA, DSA, ECDH, ECDSA, Gost 34.10 (2001), Nyberg-Rueppel, and Rabin-Williams Signature schemes.

5.2.8.1.3.6 borZoi

The borZoi library is an ECC library, designed for ease of use and a minimum risk of security problems due to incorrect use. There are C++ and Java libraries, and a Java Hyper elliptic curve lib.

The borZoi implements the following algorithms using elliptic curves defined over finite fields of characteristic 2 (GF2m):

- ECDSA (Elliptic Curve Digital Signature Algorithm) as specified in ANSI X9.62, FIPS 186-2 and IEEE P1363.

- ECIES (Elliptic Curve Integrated Encryption Scheme) as specified in ANSI X9.63 and the IEEE P1363a Draft.

- Elliptic Curve Diffie-Hellman Key Agreement Scheme as specified in ANSI X9.63 and IEEE P1363.

The AES symmetric encryption scheme (NIST AES draft) and SHA-1 hash algorithm (FIPS 180-1) are also included.

5.2.8.1.3.7 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).

Crypto++ public key support includes several ECC algorithms and schemes like Elliptic Curve Diffie-Hellman Key Agreement (ECDH), Elliptic Curve Menezes-Qu-Vanstone Key Agreement (ECMQV), Elliptic Curve Integrated Encryption Scheme (ECIES), Elliptic Curve Digital Signature Algorithm (ECDSA) and Elliptic Curve Nyberg Rueppel Signature Scheme (ECNR).

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.

5.2.8.1.3.8 FlexiProvider

The FlexiProvider is a powerful toolkit for the Java Cryptography Architecture (JCA/JCE). It provides cryptographic modules that can be plugged into every application that is built on top of the JCA.

The goal of the FlexiProvider project is to supply fast and secure implementations of cryptographic algorithms which are easy to use even for developers starting to work in the field of cryptography.

The included ECProvider is the provider for cryptographic algorithms which are based on elliptic curves. The provider includes the digital signature schemes ECDSA and ECNR, the Elliptic Curve Diffie-Hellman (ECDH) key agreement scheme, and the Elliptic Curve Integrated Encryption Scheme (ECIES).

The FlexiProvider has been developed by the Theoretical Computer Science Research Group of Prof. Dr. Johannes Buchmann at the Department of Computer Science at Technische Universität Darmstadt, Germany.

5.2.8.1.3.9 GMP

GMP is a free library for arbitrary precision arithmetic, operating on signed integers, rational numbers, and floating point numbers. There is no practical limit to the precision except the ones implied by the available memory in the machine GMP runs on. GMP has a rich set of functions, and the functions have a regular interface.

The main target applications for GMP are cryptography applications and research, Internet security applications, algebra systems, computational algebra research, etc.

GMP is carefully designed to be as fast as possible, both for small operands and for huge operands. The speed is achieved by using full-words as the basic arithmetic type, by using fast algorithms, with highly optimized assembly code for the most common inner loops for a lot of CPUs, and by a general emphasis on speed.

GMP is faster than any other bignum library. The advantage for GMP increases with the operand sizes for many operations, since GMP uses asymptotically faster algorithms.

5.2.8.1.3.10 KeyCzar

KeyCzar is an open source cryptographic toolkit designed to make it easier and safer for developers to use cryptography in their applications. KeyCzar supports authentication and encryption with both symmetric and asymmetric keys. Some features of KeyCzar include:

- A simple API
- Key rotation and versioning
- Safe default algorithms, modes, and key lengths
- Automated generation of initialization vectors and ciphertext signatures
- Java, Python, and C++ implementations

5.2.8.1.3.11 LiDIA

LiDIA is a C++ library for computational number theory which provides a collection of highly optimized implementations of various multiple precision data types and time-intensive algorithms. In particular, the library contains algorithms for factoring and for point counting on elliptic curves. The developer is the LiDIA Group at the Darmstadt University of Technology (Germany).

5.2.8.1.3.12 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.

5.2.8.1.3.13 Network Security Services (NSS)

Network Security Services (NSS) is a set of libraries designed to support cross-platform development of security-enabled client and server applications. Applications built with NSS can support SSL v2 and v3, TLS, PKCS #5, PKCS #7, PKCS #11, PKCS #12, S/MIME, X.509 v3 certificates, and other security standards.

NSS is available under the Mozilla Public License, the GNU General Public License, and the GNU Lesser General Public License.

The NSS software crypto module has been validated three times for conformance to FIPS 140 at Security Levels 1 and 2. The NSS libraries passed the NISCC TLS/SSL and S/MIME test suites (1.6 million test cases of invalid input data).

5.2.8.1.3.14 MatrixSSL

PeerSec Networks MatrixSSL™ is an embedded SSL and TLS implementation designed for small footprint applications and devices. PeerSec Networks offers a fully supported, commercial version as well as an open source version that is available for download. PeerSec MatrixSSL allows secure management of remote devices. Several secure embedded Web servers also use MatrixSSL for their encryption layer.

Before developing our own Secure Sockets Layer, we looked for a small, open source SSL/TLS implementation. This proved very difficult to find. We found several past attempts at an “OpenSSL Lite”, “small OpenSSL” or “embedded OpenSSL”, but none reduced the code to levels we required. The standard OpenSSL library is over 1 MB, and the best we found was more than half that. OpenSSL is a decent solution, but embedded security is one area where there was room for improvement.

MatrixSSL is available under a “dual license” model. Under this model users may choose to use MatrixSSL under the GNU General Public License or under a commercial license. In order to use the GNU version of MatrixSSL your product must be 100% GPL compliant, which includes making the source of the entire application available to the public.

Alternatively, the developer may choose to purchase a commercial version from PeerSec Networks allowing the use of MatrixSSL in a proprietary product. Commercially licensed customers get a supported product from PeerSec Networks with additional features and are free from the requirement to make their application open source.

5.2.8.1.3.15 NTL: A Library for doing Number Theory

NTL is a high-performance portable C++ library providing data structures and algorithms for arbitrary length integers; for vectors, matrices, and polynomials over the integers and over finite fields; and for

arbitrary precision floating point arithmetic. In particular, the library contains state-of-the-art implementations for lattice basis reduction. NTL is maintained by Victor Shoup.

5.2.8.1.3.16 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.

5.2.8.1.3.17 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.

5.2.8.1.3.18 Libmcrypt and MCrypt

MCrypt is a replacement for the old crypt() package and crypt(1) command, with extensions. It allows developers to use a wide range of encryption functions, without making drastic changes to their code.

The companion to MCrypt is Libmcrypt, which contains the actual encryption functions themselves, and provides a standardized mechanism for accessing them.

Any algorithm found to be covered by IP that conflicts with the GPL cannot be a part of mcryptlib as standard.

5.2.8.1.3.19 OpenSSL

The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general purpose cryptography library. The project is managed by a worldwide community of volunteers that use the Internet to communicate, plan, and develop the OpenSSL toolkit and its related documentation.

OpenSSL is based on the SSLeay library developed by Eric A. Young and Tim J. Hudson. The OpenSSL toolkit is licensed under an Apache-style license, which basically means that you are free to get and use it for commercial and non-commercial purposes subject to some simple license conditions.

5.2.8.1.3.20 PKIF: The Public Key Infrastructure (PKI) Framework.

The PKIF (Public Key Infrastructure Framework) helps to build robust, secure, correct certification path building and validation into your applications. With PKIF, your applications can use the secure infrastructure you've already built to create and verify digital signatures, authenticate users and encrypt data.

PKIF is a full-featured, standards compliant PKI enablement library. Its goal is to make it easy for your applications to take advantage of your PKI. PKIF runs on Windows and UNIX systems and is written in C++ with bindings for C# (and COM/.Net) and java. PKIF can validate certificates, create and verify signatures, encrypt and decrypt data, and much more.

5.2.8.1.3.21 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.

5.2.8.1.3.22 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.

5.2.8.1.3.23 Sokrates and the CACE Compiler

The CACE WP3 Zero-Knowledge Proof of Knowledge Compiler allows translating abstract high-level specifications of proof goals into sound implementations in C. Further, it offers the possibility to obtain an acrobat pdf file describing the protocol for documentation reasons. Finally, a pseudo-code in PIL (Protocol Implementation Language) can be output. This eases the implementation in other languages than C. The structure of the input files (PSL - Protocol Specification Language) is also explained.

The C-backend uses the GNU Multiple Precision Arithmetic Library GMP.

5.2.8.1.3.24 Bouncy Castle

The Bouncy Castle Crypto APIs is a powerful toolkit for the Java Cryptography Architecture (JCA/JCE) and a cryptographic provider for .Net framework. It provides cryptographic modules that can be plugged into every application that is built on top of the JCA.(using Java) or developed for .Net Framework (using C#, Vb.Net or other managed language).

The Bouncy Castle consists of the following:

- A lightweight cryptography API for Java and C#.
- A provider for the Java Cryptography Extension and the Java Cryptography Architecture.
- A clean room implementation of the JCE 1.2.1.
- A library for reading and writing encoded ASN.1 objects.
- A light weight client-side TLS API.
- Several generators:
    - Generators for Version 1 and Version 3 X.509 certificates, Version 2 CRLs, and PKCS12 files.

  - Generators for Version 2 X.509 attribute certificates.
  - Generators/Processors for S/MIME and CMS (PKCS7/RFC 3852).
  - Generators/Processors for OCSP (RFC 2560).
  - Generators/Processors for TSP (RFC 3161 & RFC 5544).
  - Generators/Processors for CMP and CRMF (RFC 4210 & RFC 4211).
  - Generators/Processors for OpenPGP (RFC 2440).

- A signed jar version suitable for JDK 1.4-1.6 and the Sun JCE.

The lightweight Java API works with everything from the J2ME to the JDK 1.6 and there is also an API in C# providing equivalent functionality for most of the above.

5.2.8.1.3.25 Gnu Crypto

GNU Crypto, part of the GNU project, aims at providing free, versatile, high-quality, and provably correct implementations of cryptographic primitives and tools in the Java programming language for use by programmers and end-users.

GNU Crypto is licensed under the terms of the GNU General Public License, with the "library exception" which permits its use as a library in conjunction with non-Free software. The effect of that license is similar to using the LGPL, except that static linking is permitted. GPL with that exception is sometimes called the Guile License, because the Guile implementation of Scheme (for embedding) uses this license.

5.2.8.1.3.26 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.

5.2.8.1.3.27 TinyPBC

TinyPBC is based on the RELIC Toolkit and supports MICA2 and MICAz motes. All optimization targets the ATmega128L processor. It is written in both NesC language, compatible with TinyOS, and C only so that it can be run in AVRStudio or Avrora simulators as well.

5.2.8.1.3.28 WM-ECC

WM-ECC is an Elliptic Curve Cryptography (ECC) primitive suite developed exclusively for wireless sensor motes. WM-ECC is an ECC public key primitive module on TinyOS supporting both MICAz, Tmote Sky and TeslosB motes. WM-ECC supports all ECC operations, including point addition, point doublding and point multiplication. Meanwhile, our ECDSA interface provides convenient message signature generation and verification.

WM-ECC is composed of following three modules: Bint, ECC, and ECDSA. Bint module provides all types of subroutines for large interger operations. The Bint module has been intensively optimized to offer the best performance as possible. Based on Bint module, ECC module implements all ECC operations. ECDSA is the module provides high level ECC signature generation and verification. The ECC module has to be initialized. The modular coding style in WM-ECC gives the users the great convenience to customize WM-ECC module for their own applications. For example, the Bint module can be re-used for developing RSA or DH primitives.

The last WM-ECC version only supports the 160-bit secp160r1 curve under tinyos 1.1.11.

5.2.8.1.3.29 MiniSec

MiniSec is a secure network layer that obtains the best of both worlds: low energy consumption and high security. MiniSec has two operating modes, one tailored for single-source communication, and another tailored for multi-source broadcast communication. The latter does not require per-sender state for replay protection and thus scales to large networks. MiniSec present a publicly available implementation for the Telos platform with re-deployed symmetric keys, Confidentiality, Replay protection using CBC-CS mode and Skipjack for Authentication.

5.2.8.1.4 Results

In the next two sections the most relevant results for the power node running embedded Linux and the low cost Telosb node are presented.

5.2.8.1.4.1 Power node Linux Results

The power node used in the tests where a Dell Vostro 1015 without battery and with power saving mode disable as previously referred in the second scenario methodology section. The power node may be in constant communication with other nodes, store and transfer large amounts of encrypted data with the control centre as opposed to the sporadic communication in the low power nodes. Also any denial of service ability to consume all power is limited by the multiple sources connected to the power node. The results are presented in two groups:

- Some selected ciphers from the most representative C and C++ cryptographic libraries (Botan, Crypto++, libgcrypt, libmcrypt, Nettle, OpenSSL and TomCrypt).

- Comparative performance of some cryptographic libraries (the previous group) for some selected ciphers.

The selected C and C++ cryptographic libraries are Botan, Crypto++, libgcrypt, libmcrypt, Nettle, OpenSSL and TomCrypt. The selected symmetric ciphers are the Rijndael AES, Serpent, Twofish, Blowfish, Camellia, CAST5, CAST6, xTEA and 3DES. The Rijndael AES and 3DES provide a good baseline for performance for the other ciphers. The selected asymmetric schemes presented latter in this section are ECIES, ECDSA, ECDHC, ECMQVC, DSA and RSA. We don't include more libraries, ciphers and schemes in the report to prevent visual pollution of the graphics and addition of non-relevant information.

The performance data of the different ciphers provided by each library is presented, starting with Botan. In the x axis, the length of the encrypted buffer is presented, ranging from 16 bytes to one megabyte. The buffer length is always plotted logarithmically, to show the small length in detail. In the y axis, the presented speed is the result of the division of the encrypted data length by the average absolute time in megabyte per second.

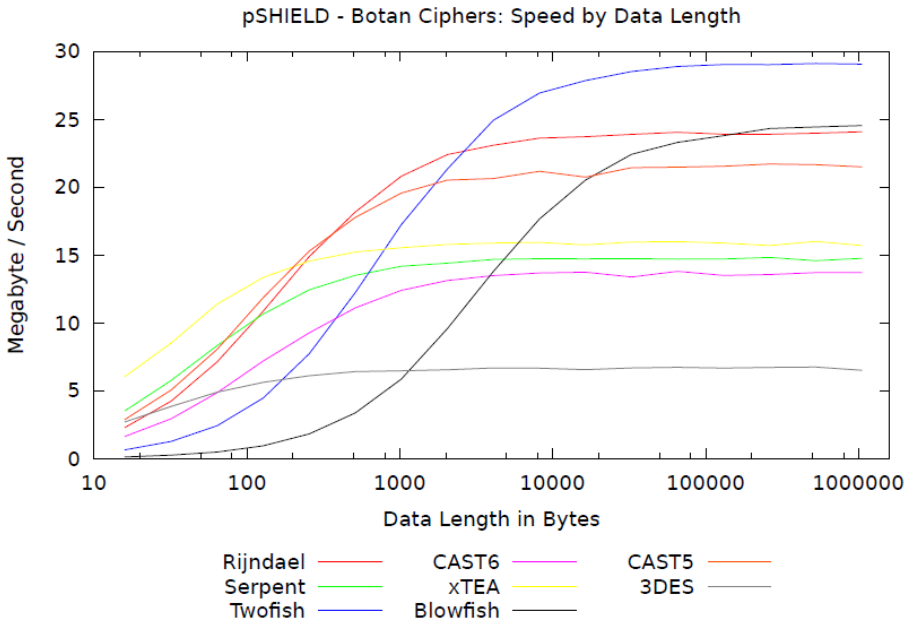


Figure 5-14: Botan speed by data length

In the previous graphic, the XTEA, the Rijndael AES and the Twofish are the best performance ciphers of the Botan library. All lines have the horizontal tendency, when the library initialization, key setup and additional overheads are compensated by the large buffers and data streams.

The Twofish and the Blowfish have a slow key setup that clearly takes more time than other ciphers, so the two curves reach the almost horizontal sloping later, with larges buffers.

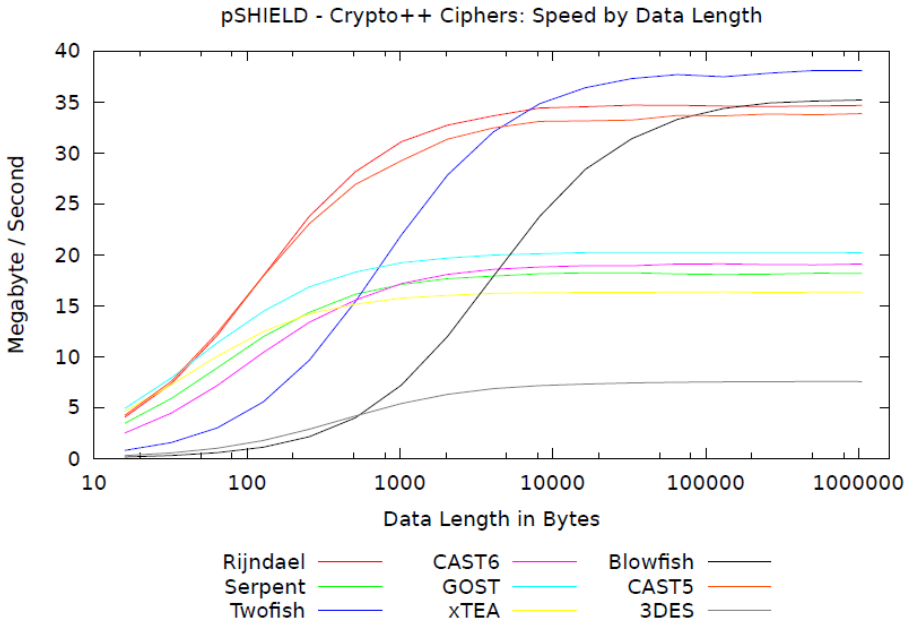


Figure 5-15: Crypto++ speed by data length

The Crypto++ best ciphers (near the top) are Gost (in the beginning, for small data lengths), CAST5 (for a very small part), AES Rijndael and Twofish (at the final part, for large data lengths). In fact, AES Rijndael and Twofish perform together almost always the best symmetric cipher response from crypto++. Other symmetric ciphers where not presented to avoid visual confusion of the graphic.

All ciphers perform significantly better than previous (Botan) and following (like Libgcrypt) implementations, as the y axis scale reach the 40 Mb/s.

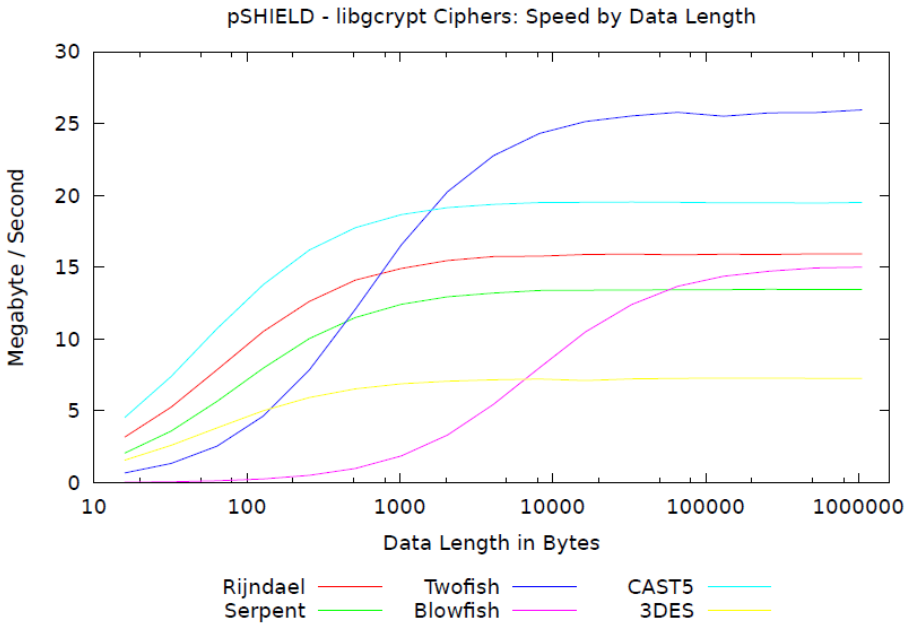


Figure 5-16: libgcrypt speed by data length

The libgcrypt CAST5 implementation outperforms the AES Rijndael all the times. The CAST5 and Twofish are the best Libgcrypt implementations. The Twofish become fastest cipher once buffers are larger than about 9000 bytes. It achieves more than 20 MB/s throughput. The start-up overhead mainly consists of cipher key-schedule context pre-calculations, but also include the library-overhead, memory-allocation and initialization.

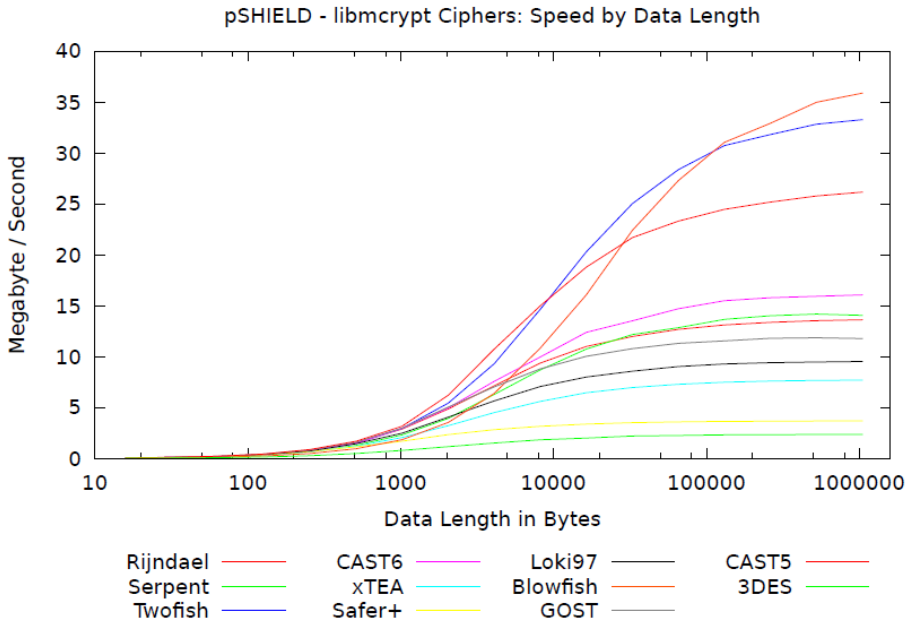


Figure 5-17: libmcrypt speed by data length

In the previous graphic, the y axis grow to 40 Megabytes/second since the CAST5, Twofish and AES Rijndael libmcrypt implementations have good performances.

The nettle library contains well-performing implementation of the most common ciphers. The best are AES Rijndael, Blowfish and Twofish. We used Nettle 2.1 and the last trunk version from the official CVS. Nettle depends on libhogweed and libhogweed depends on GMP for public key cryptography.

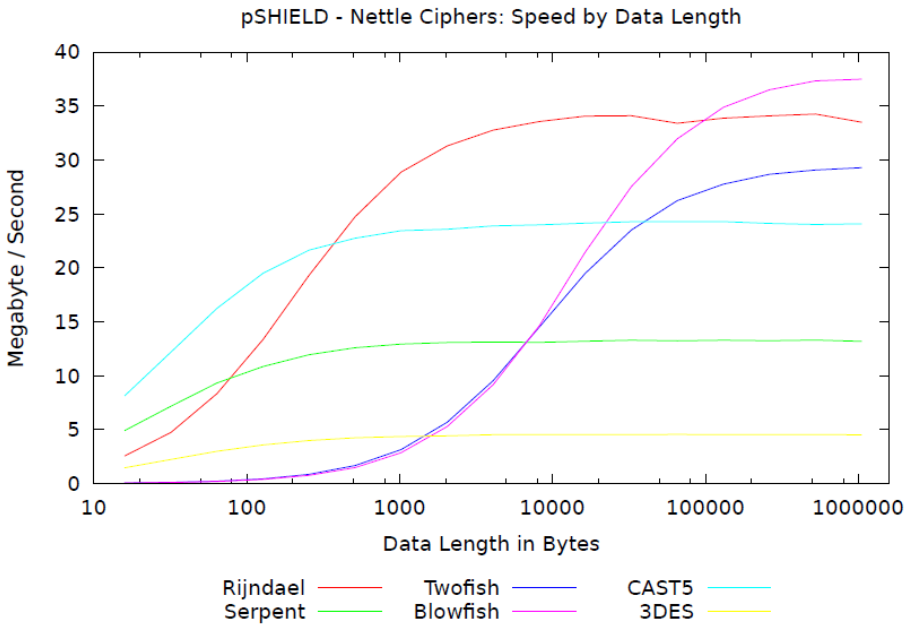


Figure 5-18: Nettle speed by data length

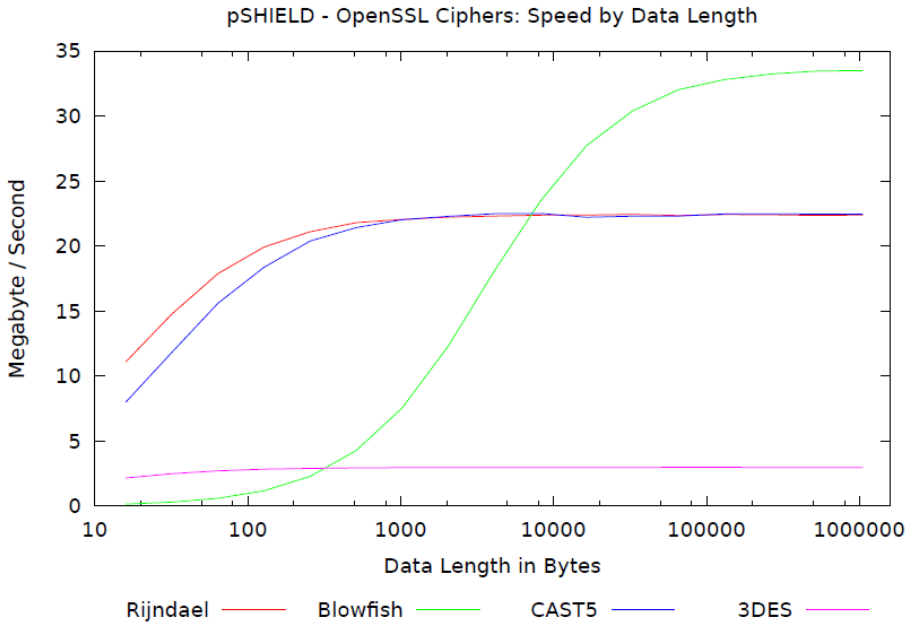


Figure 5-19: OpenSSL speed by data length

In the previous graphic, the OpenSSL best ciphers are AES Rijndael, CAST5 and Blowfish. Other algorithms like Twofish, xTEA or Serpent don't have a OpenSSL implementation.

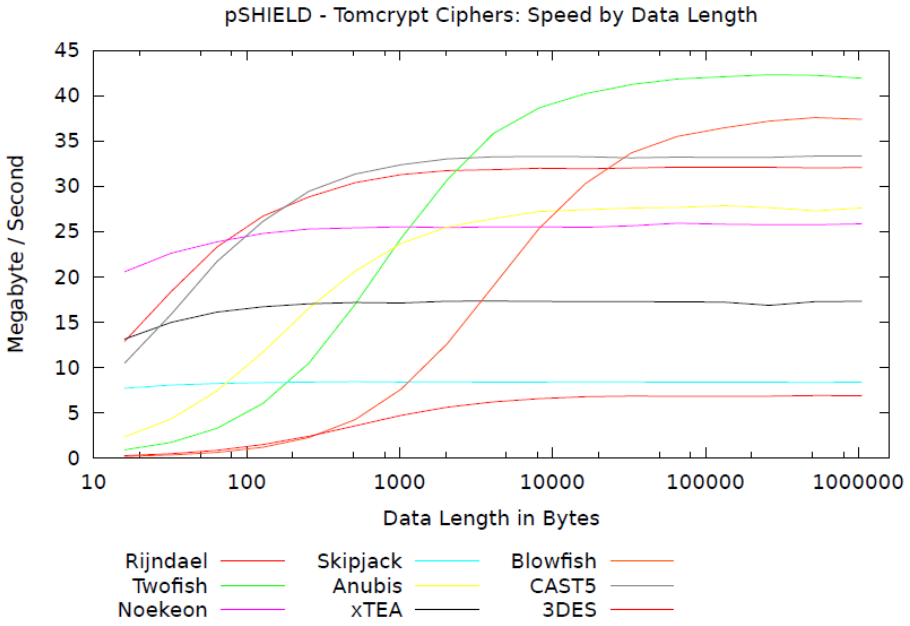


Figure 5-20: TomCrypt speed by data length

In the previous graphic, the TomCrypt best ciphers are Noekeon, AES Rijndael, CAST5 and Twofish.

Implementation benchmark

The following graphics present the performance of the implementation of the same selected algorithm by several cryptographic libraries.

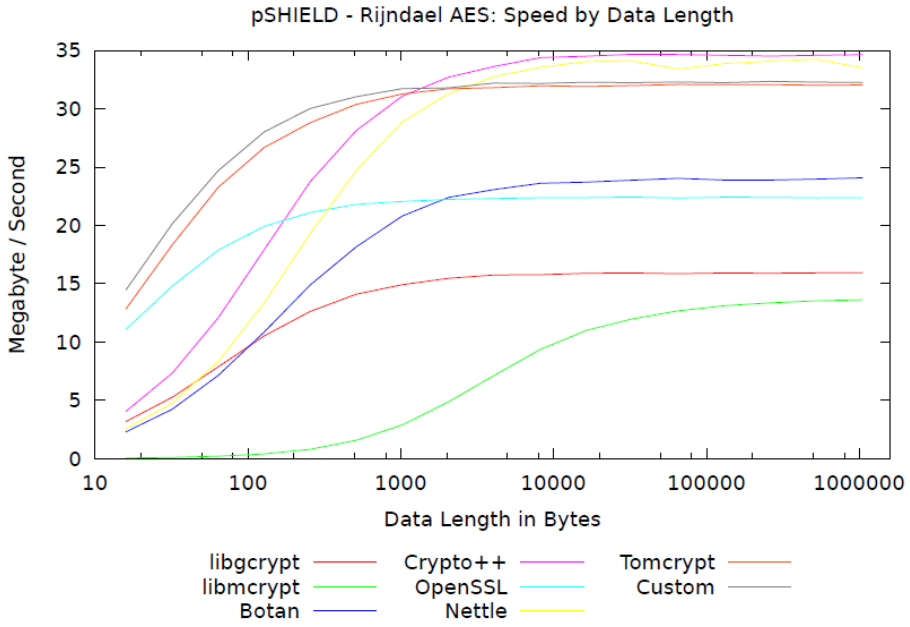


Figure 5-21: Rijndael AES speed by data length

The Rijndael AES is implemented in almost every library. The best implementations are a public domain implementation from Vincent Rijmen, Antoon Bosselaers and Paulo Barreto and crypto++. Libgcrypt and Nettle also present good performance records.

Algorithm		MiB/Second	Cycles Per Byte	Microseconds to Setup Key and IV	Cycles to Setup Key and IV
AES/GCM	**(2K tables)**	126	15.9	2.229	4680
	(64K tables)	129	15.5	9.884	20757
AES/CCM		73	27.4	0.708	1487
AES/EAX		73	27.5	1.408	2957
AES/CTR	**128-bit key**	165	12.2	0.568	1193
	192-bit key	135	14.8	0.557	1171
	256-bit key	113	17.7	0.587	1233
AES/CBC	**128-bit key**	131	15.3	0.461	969
	192-bit key	110	18.3	0.438	920
	256-bit key	97	20.6	0.457	960
AES/OFB	**128-bit key**	124	16.2	0.554	1164
AES/CFB	**128-bit key**	131	15.3	0.749	1572
AES/ECB	**128-bit key**	132	15.2	0.193	406

Table 5-7: crypto++ Rijndael AES performance under different code block modes

The performance values and the relative order of the most performance libraries may change with different code block modes. In the previous table, there is significant variation inside crypto++ implementations. The table is an extension of Table 2 with key setup costs and more data.

Also several Intel, Advanced Micro Devices (AMD) and Via processors implement extensions what greatly improve the Rijndael AES performance, as referred in section 6.2.2.2.

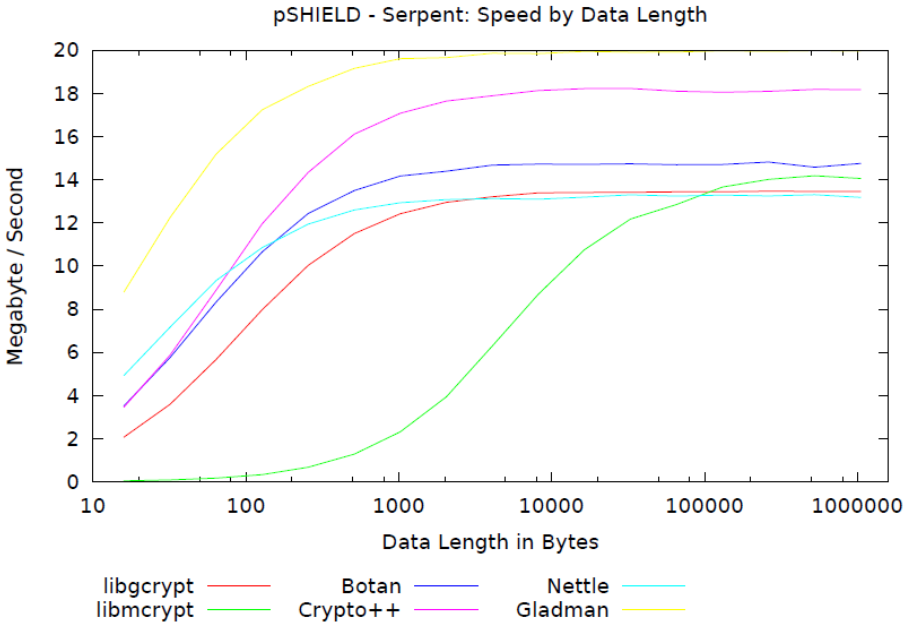


Figure 5-22: Serpent speed by data length

The best Serpent implementation was Gladman, followed by crypto++ implementation. We also add some individual implementations to the cryptographic suits to add additional information.

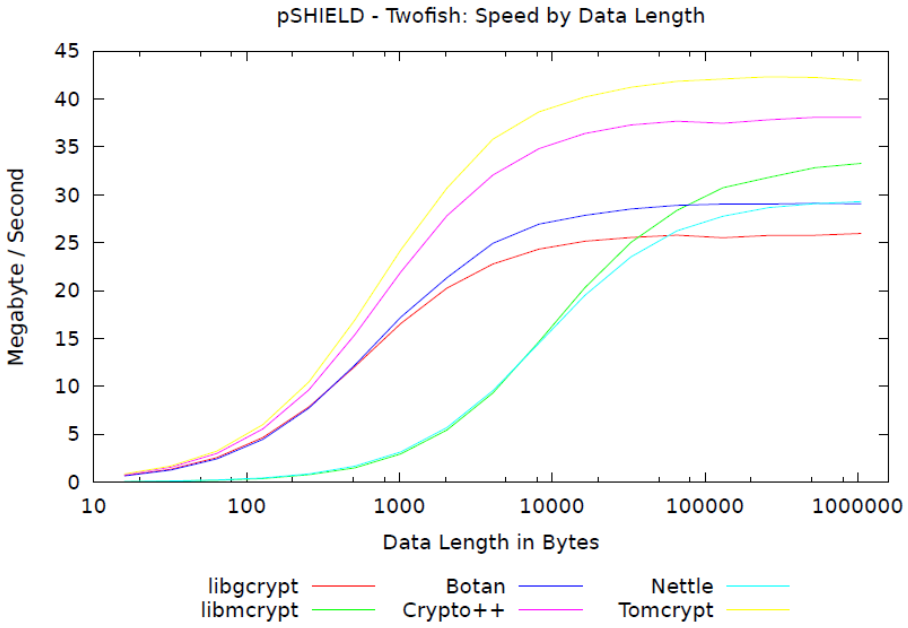


Figure 5-23: Twofish speed by data length

The best Twofish implementation is from TomCrypt, followed by crypto++. Some libraries like OpenSSL don't implement Twofish.

The public key implementations where also analysed and are presented in the next figures.

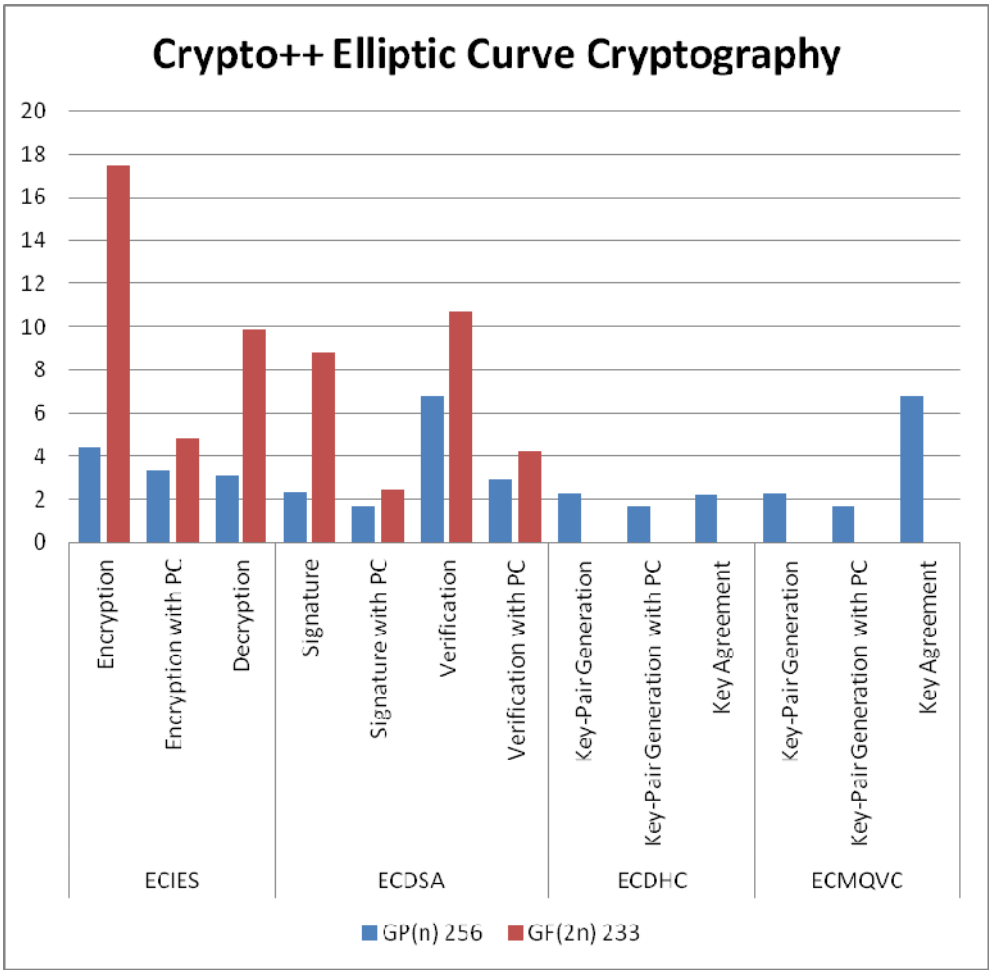


Figure 5-24: Crypto++ Elliptic Curve Ciphers Operations (Milliseconds)

In the previous figure, we present four elliptic curve schemes implemented by crypto++: ECIES, ECDSA, ECDHC and ECMQVC with two domain fields. As expected, the Galois Field GF(2^n) with field length size of 233 bit presented a worst performance (more time per operation). Also, we advise ECDHC for key exchange, since it performs better or equal than ECMQVC. The next table presents part of the data used in the previous graphic.

Algorithm	Operation	Milliseconds/Operation	Megacycles/Operation
ECIES	Encryption	4.42	9.28
	Encryption with PC	3.33	6.99
	Decryption	3.08	6.47
ECDSA	Signature	2.32	4.87
	Signature with PC	1.71	3.60
	Verification	6.76	14.20

	Verification with PC	2.91	6.12
ECDHC	Key-Pair Generation	2.24	4.71
	Key-Pair Generation with PC	1.69	3.54
	Key Agreement	2.18	4.58
ECMQVC	Key-Pair Generation	2.24	4.70
	Key-Pair Generation with PC	1.69	3.55
	Key Agreement	6.81	14.31

Table 5-8: crypto++ ECC Algorithms with GF(p) 255 Domain Field

In the next figure, some public key ciphers with similar security level, implemented in Libmcrypt 2.5.8 are presented. The RSA with key length of 1024, 2048, 3072 and 4096; DSA with 160 block length and 1024 key length; DSA with 224 block length and 2048 key length; DSA with 256 block length and 3072 key length; ECDSA with GP(p) 256 and 192, 224, 256, 384 and 521 bits key length.

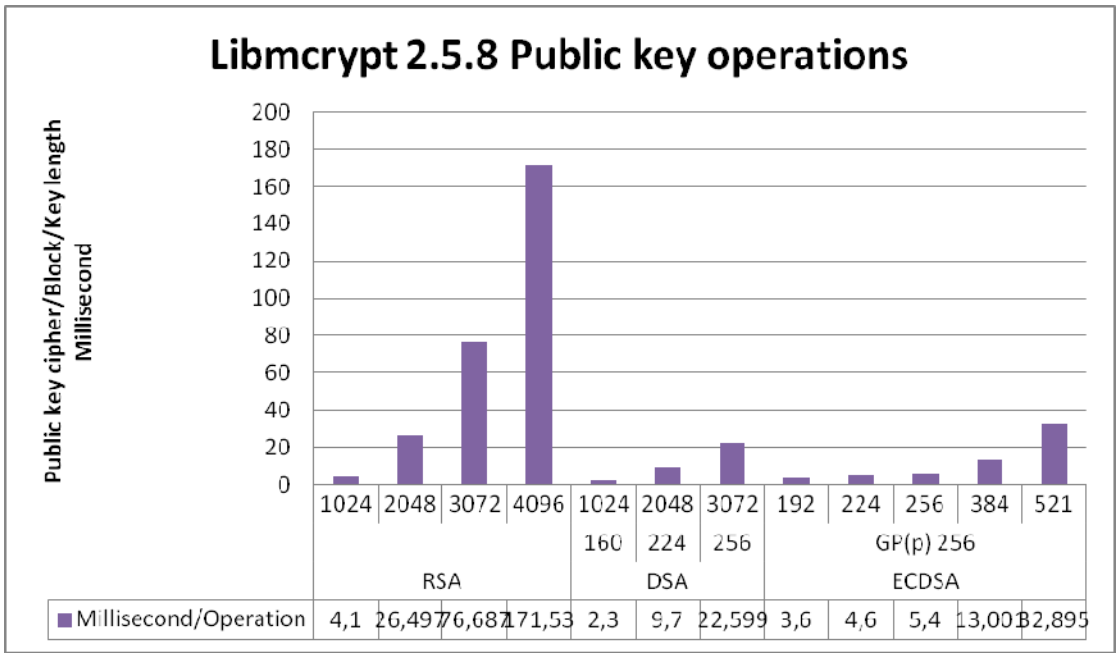


Figure 5-25: Libmcrypt 2.5.8 RSA, DSA and ECDSA Operations

The RSA starts with a good performance for low security levels, but the performance exponentially deteriorates. The RSA and DSA are behind ECDSA and other elliptic curve schemes starting from small ECC key lengths are more strong (with more security strength) than larger RSA and DSA key lengths as the next table taken from RFC 4492, illustrate.

Symmetric	ECC	DH / DSA / RSA
80	163	1024
112	233	2048
128	283	3072
192	409	7680
256	571	15360

Table 5-9: RFC 4492 Comparable Key Sizes (in bits)

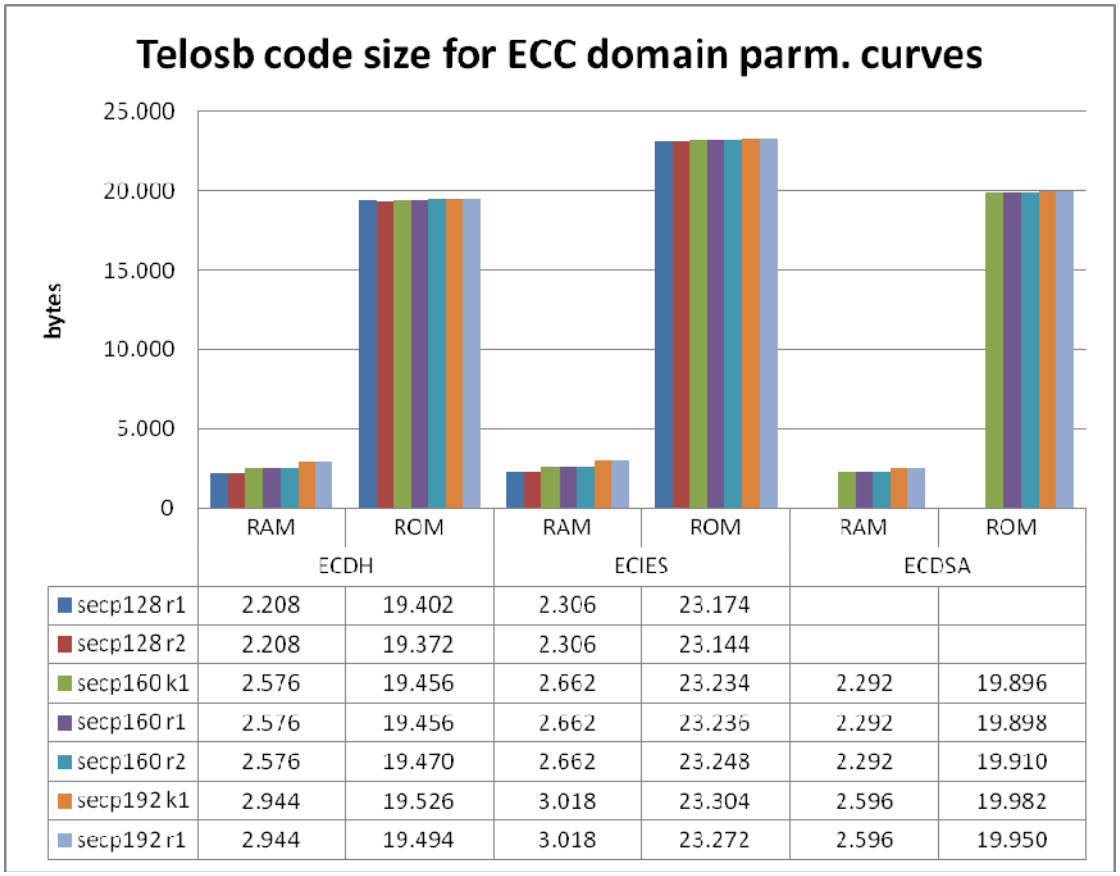
The Lenstra and Verheul Equations, published in 2001 [103] and updated in 2004 [104] are a substantial mark in the relation between the key length and the security strength.

In the US, the NIST Recommendations (2007) and the NSA Suite B Cryptography [105] proposed in 2010 are connected with several RFC (like RFC 3766). In Europe, the French Network and Information Security Agency (FNISA) recommendations (in 2010) and German BSI Recommendations (in 2011) also suggest or imposed the minimal security level and relate the expected timeframe for digital document protection and the key length.

In addition, several academic papers study this relation. For example, the RSA key length size recommended for today usage should be at least 2048 bits as opposed to ECC's 224 bits [17].

5.2.8.1.4.2 Low cost Telosb node Results

We evaluated Telosb without the use of self-tests at start-up, with several objectives: maximize performance and the code minimizing objective. For the first objective, we used all optimization, at the cost of some additional size. For the second objective, we turned off all optimizations, achieving the minimal code size and then turned on each optimization using the Makefile flags, ordering the optimization by performance increase. All the following results are taken without any radio communication since the radio transmission and processing should be measured in a different phase from the cryptographic functions.



	ECDH RAM	ECDH ROM	ECIES RAM	ECIES ROM	ECDSA RAM	ECDSA ROM
secp128r1	2.208	19.402	2.306	23.174		
secp128r2	2.208	19.372	2.306	23.144		
secp160k1	2.576	19.456	2.662	23.234	2.292	19.896
secp160r1	2.576	19.456	2.662	23.236	2.292	19.898
secp160r2	2.576	19.470	2.662	23.248	2.292	19.910
secp192k1	2.944	19.526	3.018	23.304	2.596	19.982
secp192r1	2.944	19.494	3.018	23.272	2.596	19.950

Figure 5-26: Telosb code size for different SECP elliptic curve domain parameters

As expected, as the security strength and the associated field length size rises from 128 to 192 bits, so did the code size slightly increase. In the next figure, the Ecdsa operation times in the Telosb motes for each Ecc recommended elliptic curve domain parameters are presented.

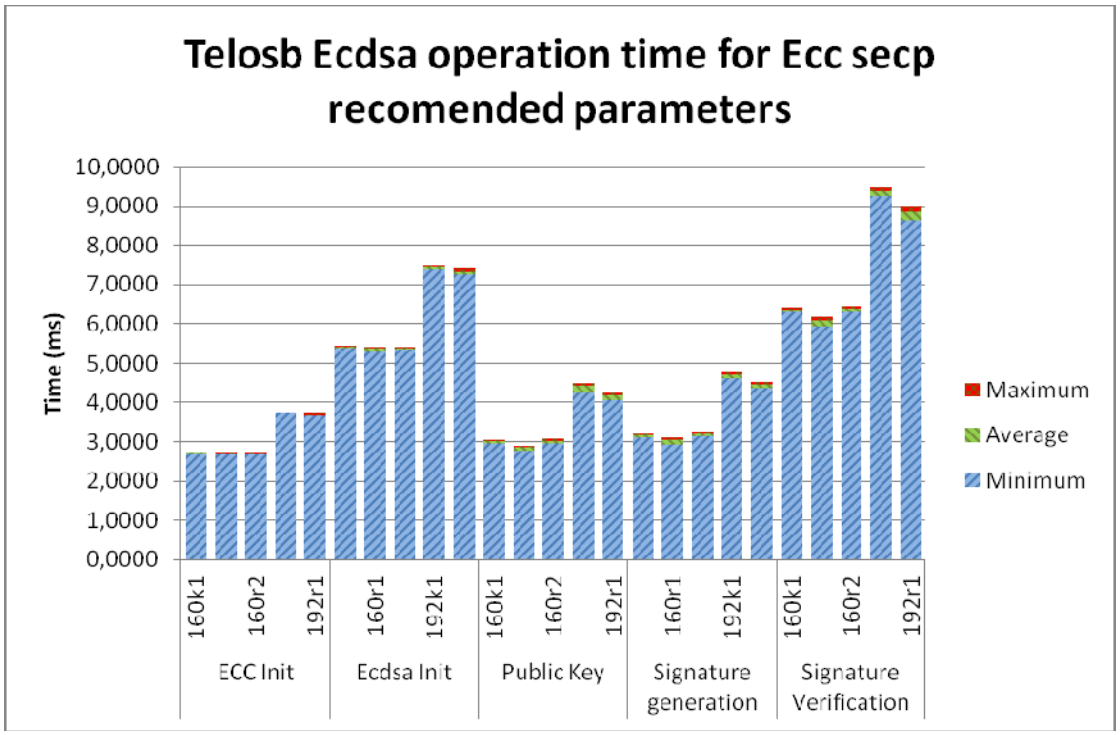


Figure 5-27: ECDSA Telosb operation time for SEC recommended elliptic curve domain parameters

As expected, as the associated field length size rises from 128 to 192 bits, so did the Ecdsa operation time increase. We can notice that Koblitz elliptic curves like spec192k1 have worst performance than similar random chosen domain parameters (like spec192r1).

The Ecdsa times for the curves spec128r1 and spec128r2 were removed because the system on the motes do not yet comply with the elliptic curve standards. The spec160k1 did not comply in beginning of tests, but a bug of TinyECC was detect, correct and reported to the authors.

The Ecies operation times on the Telosb motes is presented in the following figure.

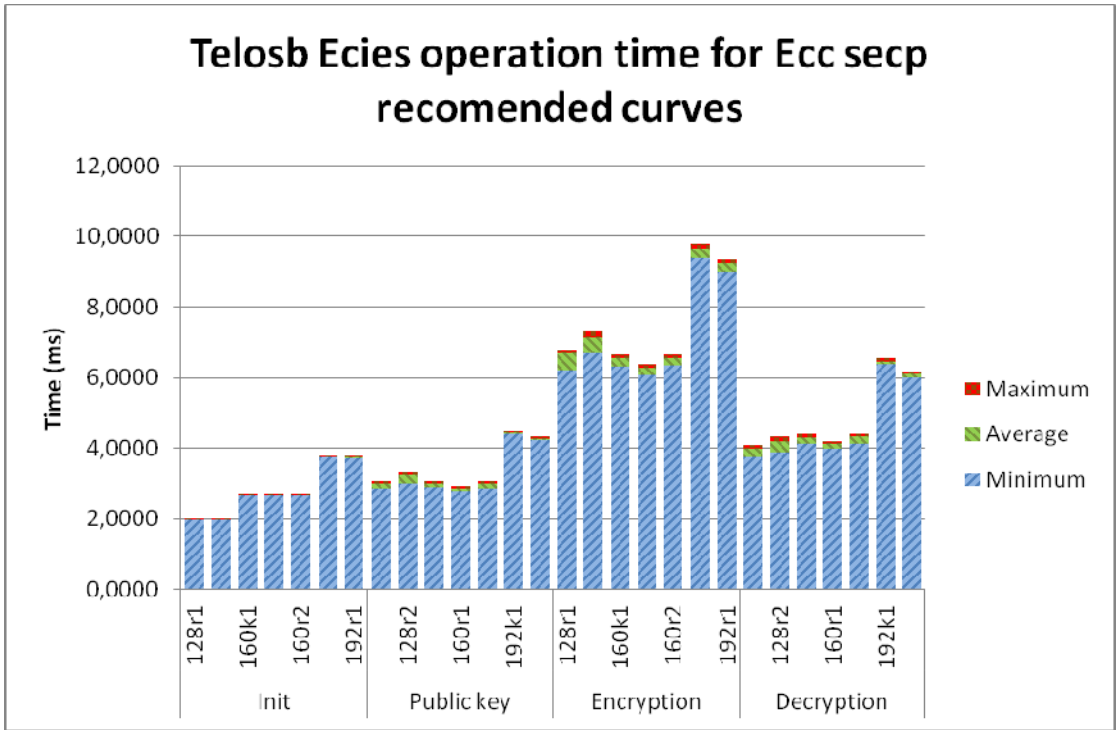


Figure 5-28: ECIES Telosb operation time by SEC recommended elliptic curve domain parameters.

We can notice a wider variation of encryption time of the spec128r1 and spec128r2 curves on Ecies. Only when the associated field length size rises from 160 to 192 bits, does Ecdsa operation time increase (public key generation, encryption and decryption).

As referred in Ecdsa, the Koblitz elliptic curves like spec192k1 have worst performance than similar random chosen domain parameters (like spec192r1) since different optimizations are apply.

In the next illustration, the Ecdh operation times in the Telosb motes is presented.

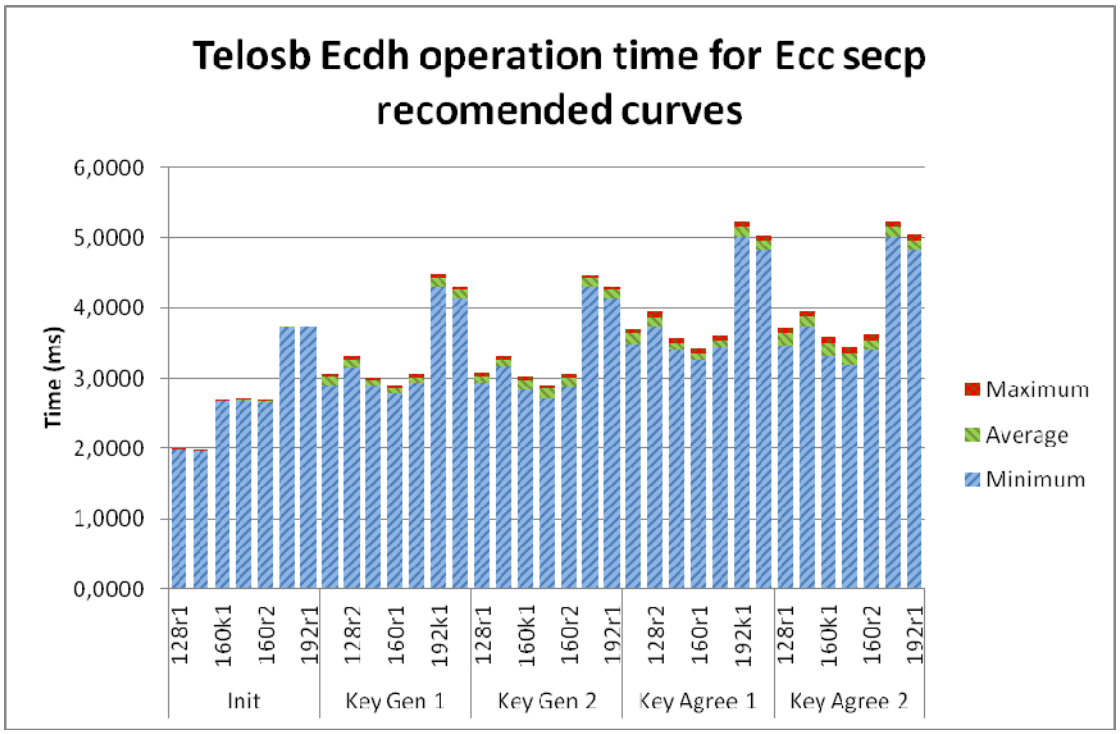


Figure 5-29: ECDH Telosb operation time for SEC recommended elliptic curve domain parameters

As in previous cases, the operation time with 128 and 160 field length size is similar, so all 128 curves are not recommended.

Other additional tests were performed. Some symmetrical ciphers were adapted and the CC2420 Radio chip was used to encrypt and decrypt but, at this stage, we cannot clearly differentiate between the radio transition overhead and the cryptography function costs.

5.2.8.1.5 Conclusions

The main objective of this evaluation is the cipher and cryptographic libraries selection for the pSHIELD scenario. This evaluation was performed with security characteristics quantification in the Cryptographic Algorithm section (5.2) and with performance characteristics measurement during the previous section (5.2.8.1.4).

A large number of symmetric ciphers have been designed to date and they vary in their security and performance characteristics. The security of a symmetric cipher cannot be easily established at design time and usually many years of exposure to public scrutiny are required in order to consider a cipher secure. On the other hand, performance characteristics can be measured and the best-performing cipher can be objectively selected.

As a result, we conclude that AES (Rijndael) and Camellia are good candidate symmetric ciphers for use in the pSHIELD middleware because of its extensive analysis by the world's best cryptographers and because of its excellent performance characteristics. Salsa is an good stream pSHIELD candidate. The asymmetric ciphers candidates for the middleware are 3 elliptic curve schemes: ECIES, ECDSA and ECDH. Every proposed ECC schemes have different application in Encryption/Decryption, Signing/Verification and Key exchange.

5.2.8.1.5.1 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 implemented by TinyECC. Every proposed ECC schemes have different application in Encryption/Decryption, Signing/Verification and Key exchange.

5.2.8.1.5.2 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 public-key ECC support is good with ECIES, ECDH and ECDSA normally implemented. Additional unusual schemes are also supported (like ECMQVC in Crypto++).

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.

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.

6 References

[1] S. McLaughlin, D. Podkuiko, and P. McDaniel, “Energy Theft in the Advanced Metering Infrastructure,” in *Critical Information Infrastructures Security*, vol. 6027, E. Rome and R. Bloomfield, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 176-187.

[2] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, “Security in embedded systems: Design challenges,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, pp. 461–491, Aug. 2004.

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Wireless sensor networks: a survey,” *Computer networks*, vol. 38, no. 4, pp. 393–422, 2002.

[4] E. A. Lee, “Cyber Physical Systems: Design Challenges,” in *2008 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, 2008, pp. 363-369.

[5] ZigBee Alliance, “ZigBee Alliance.” [Online]. Available: http://zigbee.org/. [Accessed: 04-Mar-2011]

[6] International Society of Automation, “ISA - The International Society of Automation.” [Online]. Available: http://www.isa.org/. [Accessed: 04-Mar-2011].

[7] Dash7 Alliance, “DASH7 Alliance.” [Online]. Available: http://www.dash7.org/. [Accessed: 04-Mar-2011].

[8] HART Communication Foundation, “HART Communication Foundation.” [Online]. Available: http://www.hartcomm.org/protocol/wihart/wireless_technology.html. [Accessed: 04-Mar-2011].

[9] J. Paek et al., “The Tenet architecture for tiered sensor networks,” *ACM Transactions on Sensor Networks (TOSN)*, vol. 6, pp. 34:1–34:44, Jul. 2010.

[10] A. A. Cárdenas, S. Amin, and S. Sastry, “Research challenges for the security of control systems,” in *Proceedings of the 3rd conference on Hot topics in security*, 2008, p. 6.

[11] D. Dolev and A. Yao, “On the security of public key protocols,” *Information Theory, IEEE Transactions on*, vol. 29, no. 2, pp. 198–208, 2002.

[12] NIST, “FIPS PUB 140-2.” NIST, 03-Dec-2002.

[13] M. Vuagnoux and S. Pasini, “Compromising electromagnetic emanations of wired and wireless keyboards,” in *Proceedings of the 18th conference on USENIX security symposium*, Montreal, Canada, 2009, pp. 1–16.

[14] R. J. Anderson, “Security Engineering: A guide to building dependable distributed systems,” 2008.

[15] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[16] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, “A survey of lightweight-cryptography implementations,” *IEEE Design and Test*, pp. 522–533, 2007.

[17] A. S. Wander, N. Gura, H. Eberle, V. Gupta, and S. C. Shantz, “Energy analysis of public-key cryptography for wireless sensor networks,” 2005.

[18] C. K. Koc, “RSA hardware implementation,” *RSA Laboratories, RSA Data Security, Inc*, 1996.

[19] C. K. Koc, *High-speed RSA implementation*. Technical Report, RSA Laboratories, 1994.

[20] R. Perlman, “An overview of PKI trust models,” *Network, IEEE*, vol. 13, no. 6, pp. 38–43, 1999.

[21] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer, *OpenPGP message format*. RFC 2440, November, 1998.

[22] M. SKALA, R. MICHAEL, N. HERNAEUS, R. GUYOMARCH, and K. WERNER, “GnuPG,” *Open source implementation. URL http://www. gnupg. org/. Refer to gnupg-2.0*, vol. 9, 2009.

[23] D. R. Hankerson, S. A. Vanstone, and A. J. Menezes, *Guide to elliptic curve cryptography*. Springer-Verlag New York Inc, 2004.

[24] T. ElGamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” in *Advances in Cryptology*, 1985, pp. 10–18.

[25] W. Diffie and M. Hellman, “New directions in cryptography,” *Information Theory, IEEE Transactions on*, vol. 22, no. 6, pp. 644–654, 1976.

[26] Y. Zheng and T. Matsumoto, “Breaking smart card implementations of ElGamal signature and its variants,” *rump session of Asiacrypt*, vol. 96.

[27] ISO, “ISO - International Organization for Standardization.” [Online]. Available: http://www.iso.org/iso/home.html. [Accessed: 04-Aug-2011].

[28] ANSI, “ANSI - American National Standards Institute.” [Online]. Available: http://ansi.org/. [Accessed: 04-Aug-2011].

[29] SECG, “SECG - Standards for Efficient Cryptography Group.” [Online]. Available: http://www.secg.org/. [Accessed: 04-Mar-2011].

[30] NIST, “NIST - National Institute of Standards and Technology.” [Online]. Available: http://www.nist.gov/index.html. [Accessed: 04-Aug-2011].

[31] B. Preneel et al., “NESSIE D21-Performance of Optimized Implementations of the NESSIE Primitives,” 2003.

[32] Lin Yuan and Gang Qu, “Design space exploration for energy-efficient secure sensor network,” in *The IEEE International Conference on Application-Specific Systems, Architectures and Processors, 2002. Proceedings*, 2002, pp. 88- 97.

[33] S. H. Standard, “Federal Information Processing Standard Publication# 180,” *US Department of Commerce, National Institute of Standards and Technology*, vol. 56, pp. 57–71, 1993.

[34] D. Saha, D. Mukhopadhyay, and D. RoyChowdhury, *A diagonal fault attack on the Advanced Encryption Standard.*

[35] A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich, and A. Shamir, *Key Recovery Attacks of Practical Complexity on AES Variants With Up To 10 Rounds*. 2009.

[36] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, *Performance comparison of the AES submissions*. Citeseer, 1999.

[37] "Serpent home page." [Online]. Available: http://www.cl.cam.ac.uk/~rja14/serpent.html. [Accessed: 05-Aug-2011].

[38] PGP, "The International PGP Home Page." [Online]. Available: http://www.pgpi.org/. [Accessed: 05-Aug-2011].

[39] C. Nessie, "NESSIE (The New European Schemes for Signatures, Integrity, and Encryption)," 2001.

[40] B. S. Kaliski and Y. L. Yin, "On the security of the RC5 encryption algorithm," 1998.

[41] B. Preneel et al., "Comments by the NESSIE Project on the AES Finalists," 2000.

[42] B. Preneel et al., "NESSIE security report," *Deliverable D20, NESSIE Consortium. Feb*, 2003.

[43] K. Aoki et al., "Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms—Design andAnalysis," in *Selected Areas in Cryptography*, 2001, pp. 39–56.

[44] Y. TSUNOO, H. KUBO, M. YAMADA, T. SUZAKI, and H. MIYAUCHI, "Differential and Linear cryptanalysis of CIPHERUNICORN-A.," *IEIC Technical Report (Institute of Electronics, Information and Communication Engineers)*, vol. 102, no. 212, pp. 61–68, 2002.

[45] "A Cryptographic Review of Cipherunicorn-A: cryptrec project".

[46] T. Kaneko, "Report on Evaluation of Symmetric-Key Cryptographic Techniques," 2003.

[47] T. Shimoyama et al., "The SC2000 Block Cipher," in *Proceedings of First Open NESSIE Workshop*, 2000.

[48] H. Yanami, T. Shimoyama, and O. Dunkelman, "Differential and linear cryptanalysis of a reduced-round SC2000," in *Fast Software Encryption*, 2002, pp. 639–642.

[49] H. Raddum and L. Knudsen, "A differential attack on reduced-round SC2000," in *Selected Areas in Cryptography*, 2001, pp. 190–198.

[50] O. Dunkelman and N. Keller, "Boomerang and rectangle attacks on SC2000," in *Proceedings of Second Open NESSIE Workshop*, 2001.

[51] D. Watanabe, S. Furuya, H. Yoshida, K. Takaragi, and B. Preneel, "A New Keystream Generator MUGI," in *Fast Software Encryption*, vol. 2365, J. Daemen and V. Rijmen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 179-194.

[52] S. Seys, "Cryptographic Algorithms and Protocols for Security and Privacy in Ad Hoc Networks," PhD thesis, Katholieke Universiteit Leuven, 2006.

[53] J. D. Golić, "A Weakness of the Linear Part of Stream Cipher MUGI," in *Fast Software Encryption*, vol. 3017, B. Roy and W. Meier, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 178-192.

[54] A. Biryukov and A. Shamir, "Analysis of the Non-linear Part of Mugi," in *Fast Software Encryption*, vol. 3557, H. Gilbert and H. Handschuh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 320-329.

[55] M. Henricksen and E. Dawson, "Rekeying Issues in the MUGI Stream Cipher," in *Selected Areas in Cryptography*, vol. 3897, B. Preneel and S. Tavares, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 175-188.

[56] I. P. A. Cryptrec, *Evaluation of cryptographic techniques.*

[57] E. Tews, R.-P. Weinmann, and A. Pyshkin, “Breaking 104 Bit WEP in Less Than 60 Seconds,” in *Information Security Applications*, vol. 4867, S. Kim, M. Yung, and H.-W. Lee, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 188-202.

[58] J. Lee, J. Park, S. Lee, and J. Kim, “The SEED encryption algorithm,” *SEED*, 2005.

[59] ISO - International Organization for Standardization and ISO - International Organization for Standardization, “ISO/IEC 18033-3:2010.” [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54531. [Accessed: 24-Aug-2011].

[60] J. Lee, J. Yoon, H. Lee, and S. Lee, “The SEED cipher algorithm and its use with IPsec,” *SEED*, 2005.

[61] A. Kircanski and A. M. Youssef, “Differential Fault Analysis of HC-128,” in *Progress in Cryptology – AFRICACRYPT 2010*, vol. 6055, D. J. Bernstein and T. Lange, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 261-278.

[62] E. Zenner, “A Cache Timing Analysis of HC-256,” in *Selected Areas in Cryptography*, vol. 5381, R. M. Avanzi, L. Keliher, and F. Sica, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 199-213.

[63] S. Maitra, G. Paul, S. Raizada, S. Sen, and R. Sengupta, “Some observations on HC-128,” *Designs, Codes and Cryptography*, vol. 59, no. 1-3, pp. 231-245, Dec. 2010.

[64] S. Babbage et al., “The eSTREAM portfolio,” *eSTREAM, ECRYPT Stream Cipher Project*, 2008.

[65] C. Cid, M. Robshaw, and E. I. I. D. D. SymLab, “The eSTREAM portfolio 2009 annual update,” *eSTREAM, ECRYPT Stream Cipher Project, Tech. Rep*, 2009.

[66] S. Babbage et al., “The eSTREAM portfolio,” *eSTREAM, ECRYPT Stream Cipher Project*, 2008.

[67] Y. Lu, H. Wang, and S. Ling, “Cryptanalysis of Rabbit,” *Information Security*, pp. 204–214, 2008.

[68] V. Rijmen, “Analysis of Rabbit,” *Crypromathic September*, vol. 5, 2003.

[69] Y. Lu and Y. Desmedt, “Improved Distinguishing Attack on Rabbit,” in *Information Security*, vol. 6531, M. Burmester, G. Tsudik, S. Magliveras, and I. Ilić, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 17-23.

[70] A. Kircanski and A. Youssef, “Differential Fault Analysis of Rabbit,” in *Selected Areas in Cryptography*, 2009, pp. 197–214.

[71] D. J. Bernstein, “Extending the Salsa20 nonce”.

[72] C. Berbain et al., “Sosemanuk, a Fast Software-Oriented Stream Cipher,” in *New Stream Cipher Designs*, vol. 4986, M. Robshaw and O. Billet, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 98-118.

[73] J.-K. Lee, D. H. Lee, and S. Park, “Cryptanalysis of Sosemanuk and SNOW 2.0 Using Linear Masks,” in *Advances in Cryptology - ASIACRYPT 2008*, vol. 5350, J. Pieprzyk, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 524-538.

[74] X. Feng, J. Liu, Z. Zhou, C. Wu, and D. Feng, “A Byte-Based Guess and Determine Attack on SOSEMANUK,” in *Advances in Cryptology - ASIACRYPT 2010*, vol. 6477, M. Abe, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 146-157.

[75] D. Strobel and I. C. Paar, “Side channel analysis attacks on stream ciphers,” 2009.

[76] S. Babbage and M. Dodd, “The MICKEY Stream Ciphers,” in *New Stream Cipher Designs*, vol. 4986, M. Robshaw and O. Billet, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 191-209.

[77] R. S. Mitra, P. S. Roop, and A. Basu, “An overview of MICKEY: A knowledge-based hardware-software codesign framework for microprocessor-based systems,” *Sadhana*, vol. 21, no. 6, pp. 719-739, Dec. 1996.

[78] J. Hong and W.-H. Kim, “TMD-Tradeoff and State Entropy Loss Considerations of Streamcipher MICKEY,” in *Progress in Cryptology - INDOCRYPT 2005*, vol. 3797, S. Maitra, C. E. Veni Madhavan, and R. Venkatesan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 169-182.

[79] D. Priemuth-Schmid and A. Biryukov, “Slid pairs in Salsa20 and Trivium,” *Progress in Cryptology-INDOCRYPT 2008*, pp. 1–14, 2008.

[80] D. J. Bernstein, “Which phase-3 eSTREAM ciphers provide the best software speeds?,” *Available via cr. yp. to/streamciphers/phase3speed-20080331.pdf*.

[81] J. Kim, A. Biryukov, B. Preneel, and S. Hong, “On the security of HMAC and NMAC based on HAVAL, MD4, MD5, SHA-0 and SHA-1,” *Security and Cryptography for Networks*, pp. 242–256, 2006.

[82] S. Contini and Y. Yin, “Forgery and partial key-recovery attacks on HMAC and NMAC using hash collisions,” *Advances in Cryptology–ASIACRYPT 2006*, pp. 37–53, 2006.

[83] P. A. Fouque, G. Leurent, and P. Nguyen, “Full key-recovery attacks on HMAC/NMAC-MD4 and NMAC-MD5,” *Advances in Cryptology-CRYPTO 2007*, pp. 13–30, 2007.

[84] L. Wang, K. Ohta, and N. Kunihiro, “New key-recovery attacks on HMAC/NMAC-MD4 and NMAC-MD5,” *Advances in Cryptology–EUROCRYPT 2008*, pp. 237–253, 2008.

[85] D. Forte, “The death of MD5,” *Network Security*, vol. 2009, no. 2, pp. 18–20, 2009.

[86] C. Karlof, N. Sastry, and D. Wagner, “TinySec: a link layer security architecture for wireless sensor networks,” in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, 2004, pp. 162–175.

[87] Advanced Encryption Standard (AES), ser. FIPS PUB 197, November 2001.

[88] IEEE Standard for Information Technology – Telecommunications and information Exchange between systems – Local and metropolitan area networks – Specific requirements Part 15.4. Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs), ser. IEEE Standard 802.15.4-2006, September 2006.

[89] AES 128 – A C Implementation for Encryption and Decryption. Texas Instrument, SLAA397A July 2009.

[90] Çetin K. Koç. Cryptographic Engineering.

[91] A. C. Zigiotto and R. d'Amore. A low-cost FPGA implementation of the Advanced Encryption Standard algorithm. In Proc. Symposium on Integrated Circuits and Systems Design (SBCCI'02), pages 191–196, 2002

[92] A. Satoh, S. Morioka, K. Takano, and S. Munetoh. A compact Rijndael hardware architecture with S-Box optimization.

[93] P. Chodowiec and K. Gaj. Very compact FPGA implementation of the AES algorithm. In C¸. K. Ko¸c and C. Paar, editors, Proc. International Workshop on Cryptographic Hardware and Embedded Systems (CHES'03).

[94] V. Fischer and M. Drutarovsk´y. Two methods of Rijndael implementation in reconfigurable hardware. In C¸. K. Ko¸c and C. Paar, editors, Proc. Cryptographic Hardware and Embedded Systems (CHES'01), volume 2162 of LNCS, pages 81– 96. Springer-Verlag, 2001.

[95] Kris Gaj and Pawel Chodowiec. "FPGA and ASIC Implementations of AES".

[96] Shammi Didla, aaron Ault and Saurabh Bagchi. "Optimizing AES for Embedded Devices and Wireless Sensor Networks".

[97] Institute of Electrical and Electronics Engineers, Inc., IEEE Std. 802.11i-2004, Amendment to Standard for Telecommunications and Information Exchange Between Systems – LAN/MAN Specific Requirements – Part 11: "Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Medium Access Control (MAC) Security Enhancements", July, 2004.

[98] D. Whiting, R. Housley, and N. Ferguson. "IEEE 802.11-02/001r2: AES Encryption and Authentication Using CTR Mode and CBC-MAC", March 2002.

[99] P. Rogaway and D. Wagner, "A Critique of CCM" Eprint cryptology archive, February 2003. Available at http://eprint.iacr.org.

[100] Zadia Codabux-Rossan, M. Razvi Doomun, "Performance of Interleaved Cipher Block Chaining in CCMP".

[101] D. Gay, P. Levis, R. Von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC language: A holistic approach to networked embedded systems," in *Acm Sigplan Notices*, 2003, vol. 38, pp. 1–11.

[102] D. Gay, P. Levis, D. Culler, and E. Brewer, "nesC 1.1 language reference manual," *published on nesC Website*, vol. 5, 2003.

[103] A. K. Lenstra and E. R. Verheul, "Selecting cryptographic key sizes," *Journal of cryptology*, vol. 14, no. 4, pp. 255–293, 2001.

[104] A. K. Lenstra and E. R. Verheul, "Selecting cryptographic key sizes," in *Public Key Cryptography*, 2004, pp. 446–465.

[105] N. S. A. Fact Sheet, "Suite B Cryptography," *National Security Agency Central Security Service http://www. nsa. gov/ia/industry/crypto_suite_b. Cfm*

[106] Certicom Corp. *Remarks on the security of the elliptic curve cryptosystem. A Certicom Whitepaper*, September 1997. Available from Certicom website at http://www.certicom.com/research/wecc3.html