Algorithms **2013**, *6*, 197-226; doi:10.3390/a6020197

OPEN ACCESS

algorithms

ISSN 1999-4893

www.mdpi.com/journal/algorithms

Article

Enforcing Security Mechanisms in the IP-Based Internet of Things: An Algorithmic Overview

Simone Cirani *, Gianluigi Ferrari and Luca Veltri

Department of Information Engineering, University of Parma, Parco Area delle Scienze 181/A, 43124, Parma, Italy; E-Mails: gianluigi.ferrari@unipr.it (G.F.); luca.veltri@unipr.it (L.V.)

* Author to whom correspondence should be addressed; E-Mail: simone.cirani@unipr.it; Tel.: +39-0521-905741; Fax: +39-0521-905758.

Received: 6 December 2012; in revised form: 8 February 2013 / Accepted: 7 March 2013 / Published: 2 April 2013

Abstract: The Internet of Things (IoT) refers to the Internet-like structure of billions of interconnected constrained devices, denoted as “smart objects”. Smart objects have limited capabilities, in terms of computational power and memory, and might be battery-powered devices, thus raising the need to adopt particularly energy efficient technologies. Among the most notable challenges that building interconnected smart objects brings about, there are standardization and interoperability. The use of IP has been foreseen as the standard for interoperability for smart objects. As billions of smart objects are expected to come to life and IPv4 addresses have eventually reached depletion, IPv6 has been identified as a candidate for smart-object communication. The deployment of the IoT raises many security issues coming from (i) the very nature of smart objects, e.g., the adoption of lightweight cryptographic algorithms, in terms of processing and memory requirements; and (ii) the use of standard protocols, e.g., the need to minimize the amount of data exchanged between nodes. This paper provides a detailed overview of the security challenges related to the deployment of smart objects. Security protocols at network, transport, and application layers are discussed, together with lightweight cryptographic algorithms proposed to be used instead of conventional and demanding ones, in terms of computational resources. Security aspects, such as key distribution and security bootstrapping, and application scenarios, such as secure data aggregation and service authorization, are also discussed.

Keywords: security; lightweight cryptography; Internet of Things (IoT); smart objects; secure communication protocols; secure data aggregation

1. Introduction

The Internet of Things (IoT) is an emerging concept that refers to billions of interconnected (non-human) information sources, also denoted as "smart objects", typically equipped with sensors or actuators, a tiny microprocessor, a communication interface, and a power source. Existing deployed smart objects typically use proprietary technologies, tailored to one specific application, which results in poor, if any, interoperability and thus limited diffusion on a large scale. Building interconnected and interoperable smart objects requires the adoption of standard communication protocols. International organizations, such as the Internet Engineering Task Force (IETF) and the IPSO Alliance, promote the use of the Internet Protocol (IP) as the standard for interoperability for smart objects. As billions of smart objects are expected to come to life and IPv4 addresses have eventually reached depletion, IPv6 [1] has been identified as a candidate for smart-object communication. The protocol stack that smart objects will implement will try to match classical Internet hosts in order to make it feasible to create the so-called Extended Internet, that is, the aggregation of the Internet with the IoT.

Security in the IoT scenarios is a crucial aspect that applies at different levels, ranging from technological issues to more philosophical ones, such as privacy and trust, especially in scenarios like Smart Toys. The security challenges derive from the very nature of smart objects and the use of standard protocols. In [2], the security challenges and requirements for an IP-based IoT are presented by analyzing existing Internet protocols to be applied to IoT and the limitations and problems that such a solution might introduce. In [3], the authors summarize security threats in IoT as follows:

1. cloning of smart things by untrusted manufacturers;
2. malicious substitution of smart things during installation;
3. firmware replacement attack;
4. extraction of security parameters since smart things may be physically unprotected;
5. eavesdropping attack if the communication channel is not adequately protected;
6. man-in-the-middle attack during key exchange;
7. routing attacks;
8. denial-of-service attacks;
9. privacy threats.

Threats 1, 2, 3, and 4 are related to the physical nature of smart objects, which are typically deployed in public areas and cannot be constantly supervised, thus leading to potential damages or counterfeits. Threats 5, 6, 7, and 8 are examples of security issues due to the fact that one has to deal with objects that communicate with each other. Finally, Threat 9 is related to the fact that smart objects might deal with personal or sensible data, which, if intercepted by unauthorized parties, may create ethical and privacy problems.

While it is possible to cope with physical nature-related issues only by adopting safe supplying and installation measures, such as avoiding untrusted manufacturers and installers, and by trying to protect smart objects in safe places, all other security threats can be tackled by adopting means, such as secure communication protocols and cryptographic algorithms, in order to enforce the following basic security properties:

- Confidentiality: transmitted data can be read only by the communication endpoints;

- Availability: the communication endpoints can always be reached and cannot be made inaccessible;
- Integrity: received data are not tampered with during transmission; if this does not happen, then any change can be detected;
- Authenticity: data sender can always be verified and data receivers cannot be spoofed.

There is an additional property of security that should always be taken into account: authorization. Authorization means that data can be accessed only by those allowed to do so and should be made unavailable to others. This aspect, which requires to identify the communication endpoints, is particularly relevant in those scenarios where it is necessary to ensure that private data cannot be accessed by unknown or unauthorized parties.

It is a common opinion that in the near future IP will be the base common network protocol for the IoT. This does not imply that all objects will be able to run IP. At the opposite, there will always be tiny devices, such as tiny sensors or Radio-Frequency IDentification (RFID) tags, that will be organized in closed networks implementing very simple and application-specific communication protocols and that eventually will be connected to an external network through a proper gateway. However, it is foreseen that all remaining small networked objects will exploit the benefits of IP and corresponding protocol suite.

In [4], the author tries to define the following pair of classes for constrained devices, in terms of memory capacity, in order to be used as a rough indication of device capabilities:

- class 1: RAM size = $\sim$10 KB, Flash size = $\sim$100 KB;
- class 2: RAM size = $\sim$50 KB, Flash size = $\sim$250 KB;

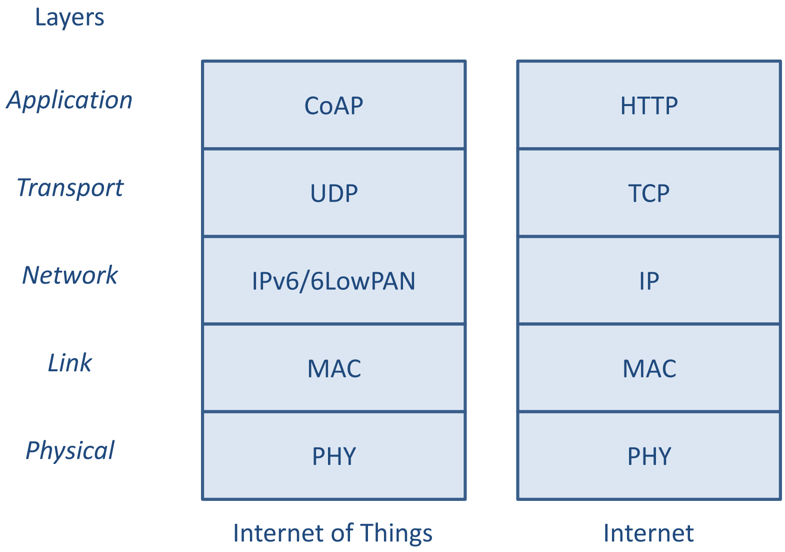
Some of these networked objects, with enough memory, computational power, and power capacity, will simply run existing IP-based protocol suite implementations. Some others will still run standard Internet protocols, but may benefit from specific implementations that try to achieve better performance in terms of memory size, computational power, and power consumption. Instead, in other constrained networked scenarios, smart objects may require additional protocols and some protocol adaptations in order to optimize Internet communications and lower memory, computational, and power requirements. There is currently a large effort within the IETF in the direction of extending existing protocols for resource-constrained networked environments. Some of the current IETF working groups targeted to these environments are: the Constrained RESTful Environments (CoRE) [5], IPv6 over Low power WPAN (6LoWPAN) [6], Routing Over Low power and Lossy networks (ROLL) [7], and the Light-Weight Implementation Guidance (LWIG) [8] working groups.

In Figure 1, a typical IP-based IoT protocol stack is depicted and compared with the classical Internet protocol stack used by standard non-constrained nodes for accessing the World Wide Web. At application layer, the HTTP [9] protocol is replaced by the Constrained Application Protocol (CoAP) [10], which is an application layer protocol to be used by resource-constrained devices, offering a REpresentational State Transfer (REST) service for Machine-to-Machine (M2M) communications, that can be easily translated to/from HTTP.

Significant reasons for proper protocol optimizations and adaptations for resource-constrained objects can be summarized as follows.

- Smart objects typically use, at physical and link layers, communication protocols (such as IEEE 802.15.4) that are characterized by small Maximum Transmission Units (MTUs), thus leading to packet fragmentation. In this case, the use of compressed protocols can significantly reduce the need for packet fragmentation and postponed transmissions.
- Processing larger packets likely leads to higher energy consumption, which can be a critical issue in battery-powered devices.
- Minimized versions of protocols (at all layers) can reduce the number of exchanged messages.

Figure 1. Comparison between the IoT and the Internet protocol stack for OSI layers.



Protocol compression is especially relevant when dealing with security protocols, which typically introduce higher overhead and increase the size of transmitted data packets. Besides protocol compression, cross-layer interaction between protocols plays a crucial role. This aspect is particularly important in order to avoid useless duplication of security features, which might have a detrimental impact on the computation and transmission performance. For instance, end-to-end security can be guaranteed by adopting IPSec at the network layer or TLS/DTLS at the transport layer. Combining these two security protocols results in a very expensive processing both at the secure channel setup phase and during packet transmission/reception.

Another important issue regarding the introduction of security protocols is interoperability. Security protocols typically allow the negotiation of some parameters to be used during operations. Such negotiations might be related to cryptographic and digital signature algorithms. In order to guarantee full interoperability among smart objects, it is necessary to define a set of mandatory options that all objects must implement for minimal support. The algorithms supported by an object are declared in a negotiation phase and a proper choice is then agreed upon by the two communicating parties. It is not necessary that the mandatory algorithms are standard algorithms used in the Internet, but can be targeted for constrained environments.

A final remark should be made about the heterogeneous nature of smart objects, whose characteristics can vary significantly with relevant differences with respect to those of conventional hosts. This means that the adoption of a suite of security protocols and cryptographic algorithms is a need and a challenge at the same time. Standardization can lead to full interoperability, yet it is extremely difficult to agree on a set of common protocols and algorithms supported by all devices.

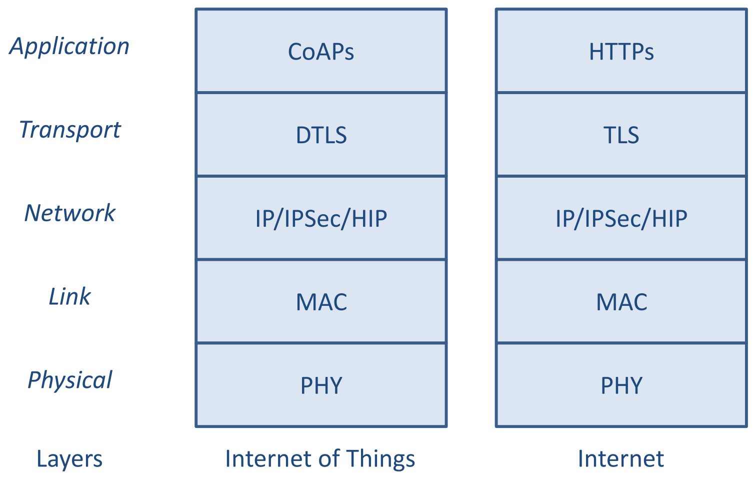
This paper aims at providing an overview of the cryptographic algorithms and security protocols suitable to deployment of smart objects. The rest of this paper is organized as follows. In Section 2, an overview of security protocols is presented. Section 3 is dedicated to the description of several lightweight cryptographic algorithms relevant for IoT scenarios. In Section 4, the focus is on key distribution and management. In Section 5, secure data aggregation is discussed. Section 6 describes the problem of authorization and protection of RESTful services in constrained environments. Finally, in Section 7 some conclusions are drawn.

2. Security Protocols

As already mentioned in Section 1, one of the most important requirements and crucial aspects for a correct deployment and diffusion of IoT is security. Several challenging security goals should be achieved, including data confidentiality, data authentication, integrity, service availability, peer entity authentication, authorization, anonymity, and/or pseudonymity. Since the protocol architecture of smart objects should adhere to the standard IP architecture (for obvious integration reasons), many of the security mechanisms already defined and currently used for the Internet can be reused in IoT scenarios. Moreover, since many of the Internet security protocols have been defined taking into account the possibility to select and properly configure the used security algorithms and other cryptographic primitives, such Internet security protocols can still be reused, possibly with proper algorithmic or configuration modifications.

In this section, the main protocols that can be used for securing IP-based end-to-end communications between smart objects are recalled, and the main issues related to this type of communications are discussed. Algorithms and other mechanisms actually used by these protocols are separately discussed next in the Section 3. According to the protocol stacks depicted in Figure 1, a direct comparison between possible layered architectures of security protocols in Internet and IoT scenarios is shown in Figure 2.

Figure 2. Comparison between the Internet and the IoT security protocols.



It is important to observe that the IoT protocol suite depicted in Figure 2 represents just the possible choices to enforce data protection (at different layers) by a smart object, rather than the actual set of security mechanisms effectively implemented and simultaneously used at different layers. At the opposite, in order to minimize the used resources, particular attention has to be devoted to avoid the repetition of the same functionalities at different layers, if not strictly required.

Referring to the IoT protocol stack of Figure 2, at the application layer there is the CoAP application protocol that can be used to interact in a request/response manner between smart objects or between a smart object and a non-constrained (standard) Internet node (possibly by using some intermediate relay/proxy node). CoAP itself does not provide primitives for authentication and data protection, so these functions should be implemented directly at the application/service layer (by directly protecting the data encapsulated and exchanged by CoAP) or at one of the underlying layers. Although data authentication, integrity, and confidentiality can be provided at lower layers, such as PHY or MAC (e.g., in IEEE 802.15.4 systems), no end-to-end security can be guaranteed without a high level of trust on intermediate nodes. However, due to the highly dynamic nature of wireless multi-hop communications expected to be used to form the routing path between remote end nodes, this kind of security (hop-by-hop) is not, in general, sufficient. For such reason, security mechanisms at network, transport, or application level should be considered instead of (or in addition to) PHY and MAC level mechanisms.

2.1. Network-Layer Security

At network layer, an IoT node can secure data exchange in a standard way by using the Internet Protocol Security (IPsec) [11]. IPSec was originally developed for IPv6, but found widespread deployment, first, as an extension in IPv4, into which it was back-engineered. IPSec was an integral part of the base IPv6 protocol suite, but has since then been made optional. IPSec can be used in protecting data flows between a pair of hosts (host-to-host communication), between a pair of security gateways (network-to-network communication), or between a security gateway and a host (network-to-host communication).

IPSec can provide confidentiality, integrity, data-origin authentication and protection against replay attacks, for each IP packet (it works at network layer). Such security services are implemented by two IPSec security protocols: Authentication Header (AH) and Encapsulated Security Payload (ESP). While the former (AH) provides integrity, data-origin authentication, and optionally anti-replay capabilities, the latter (ESP) may provide confidentiality, data-origin authentication, integrity, and anti-replay capabilities.

IPSec AH and ESP define only the way payload data (in clear or enciphered) and IPSec control information are encapsulated, while the effective algorithms for data origin authentication/integrity/confidentiality can be specified separately and selected amongst a set of available cipher suites.

This modularity makes IPSec usable also in the presence of very resource-constrained devices, if a proper algorithm that guarantees both usability and sufficient security level is selected. This means that, from an algorithmic point of view, the problem moves from the IPSec protocol itself to the actual cryptographic algorithms. Section 3 is fully dedicated to algorithm-related issues.

The keying material and the selected cryptographic algorithms used by IPSec for securing a communication are called IPSec Security Association (SA). To establish a SA, IPSec can be pre-configured (specifying a pre-shared key, hash function and encryption algorithm) or can be dynamically negotiated by the IPSec Internet Key Exchange (IKE) protocol. Unfortunately, as the IKE protocol was designed for standard Internet nodes, it uses asymmetric cryptography, which is computationally heavy for very small devices. For this reason, proper IKE extensions should be considered using lighter algorithms. These aspects are treated in Section 3.

Other problems related to the implementation of IPSec in constrained IoT nodes include data overhead (with respect to IP), configuration, and practical implementation aspects. Data overhead is introduced by the extra header encapsulation of IPSec AH and/or ESP. However, this can be limited by implementing header compression techniques, similarly to what is done in 6LoWPAN for the IP header. In [12], a possible compression mechanism for IPSec in 6LoWPAN is proposed and numerically evaluated.

Regarding practical aspects, it is worth to observe that IPSec is often designed for VPNs, thus making it difficult for them to be dynamically configurable by an application. Moreover, existing implementations are also hardly compatible with each other and often require manual configuration to interoperate.

An alternative to using IKE+IPsec is the Host Identity Protocol (HIP) [13]. The main objective of HIP is to decouple the two functions of host locators (for routing purposes) and host identifiers (for actual host identification) currently performed by IP addresses. For this purpose, HIP introduces a new namespace between IP and upper layers specific for host identification based on public cryptography. In HIP, the host identity (HI) is directly associated with a pair of public/private keys, where the private key is owned by the host and the public key is used as Host Identifier (HI). HIP defines also an Host Identity Tag (HIT), a 128-bit representation of the HI based on the hash of HI plus other information, which can be used for example as unique host identifier at the existing IPv6 API and by application protocols. HIP defines also an HIP exchange that can be used between IP hosts to establish a HIP security association that in turn can be used to start secure host-to-host communications based on the (IPSec) ESP protocol [14].

In addition to security, HIP provides methods for IP multihoming and host mobility that are important features for an IP-based IoT network architecture. Some works are also being carried on to let the HIP exchange run on very constrained devices, by using proper public-key cryptographic primitives, for example the ones described in Section 3.

2.2. Transport-Layer Security

In the current IP architecture, data exchange between application nodes can be secured at transport layer through the standard Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) protocols. TLS is the widest used secure protocol, running on top of the TCP providing to the application layer the same connection and stream-oriented interface of TCP [15]. In addition, TLS provides complete secure communication through: peer-entity authentication and key exchange (using asymmetric cryptography); data authentication, integrity, and anti-replay (through message authentication code); confidentiality (using symmetric encryption). Peer-entity authentication and key exchange is provided by the TLS Handshake phase, performed at the beginning of the communication.

DTLS, instead, has been introduced more recently in order to provide a security service similar to TLS on top of UDP [16]. Although it is still poorly supported in standard Internet nodes, it is currently the reference security protocol for IoT systems since it uses UDP as transport and does not suffer from the problems originated by the use of TCP in network-constrained scenarios (due to the extremely variable transmission delay and lossy links).

Both IPSec and DTLS provide the same security features with their own mechanisms at different stack layers. Moreover the IPSec IKE key agreement reflects almost the same DTLS Handshake function. The main advantage of securing communications at transport layer with DTLS consists in allowing more precise access control. In fact, operation at the transport layer allows applications to directly and easily select which, if any, security service has to be set up. Another practical advantage is that the adoption of DTLS can allow for the reuse of the large experience and implementations that came with TLS.

For these reasons DTLS has recently received significant attention for securing communication of constrained node/network applications and it has been standardized as the security protocol for CoAP associated to “coaps” URIs [10].

Unfortunately, there are still some few issues that should be faced in order to make DTLS more friendly for constrained devices. The most relevant ones are related to limited packet size imposed by underlying protocols such as IEEE 802.15.4. In fact, as for IPSec, DTLS introduces overhead during both handshake and data transport phases. DTLS offers fragmentation at the Handshake layer, however, this can add a significant overhead. Another solution could be to use the fragmentation offered at IPv6 or 6LoWPAN layer. Moreover, in order to reduce DTLS overhead, some packet optimization and compression mechanism can be introduced. For example, in [17] the authors propose to use the 6LoWPAN compression mechanisms for the DTLS protocol.

From the security point of view, one problem of using DTLS or IPSec is that end-to-end communication is not guaranteed when intermediate nodes such as proxies or application level gateways are introduced. In fact, both IPSec and DTLS provide secure communications at IP and transport layers respectively, and, in presence of a multi-hop application-level communications, they can assure security only within each hop. In addition, some complications in providing end-to-end security may arise also when connectivity is realized directly at IP and transport layers. There are scenarios in which a part of the network (internal) composed by constrained devices is interconnected at IP level to the rest of the (external) network, for example the Internet. Although data protection can be guaranteed through IPSec or DTLS protocols, other network attacks, like flooding or replay, may occur due to the asymmetry of the resources available at the end systems; for example a full-powered host attached to the Internet may attack a constrained device by trying to consume all power or processing resources of the limited device. In order to guarantee a proper level of protection also against this kind of attacks, an intermediate security gateway may be required at the border of the internal network. A security gateway may act as access controller, granting access to the internal network only to trusted nodes. In [18,19], the authors specifically face this issue and try to propose a solution. In particular, in case of end-to-end application level communication based on CoAP, a solution may be to require the external node to encapsulate

CoAP/DTLS/IP traffic within a proper DTLS tunnel established between the external node and the security gateway.

It is also important to note that, although DTLS provides a datagram-oriented communication service (like UDP), it establishes a point-to-point secure association that is not compatible with multicast communications (in contrast with UDP, which does support multicast). In order to make DTLS applicable in multicast IP-communication scenarios, some protocol extensions for group-key management should be introduced in the future.

2.3. Application-Layer Security

Providing security at IP layer (through IPSec) or transport layer (through TLS or DTLS) has several advantages. The main ones are: first, the same standard mechanism and the same implementation can be shared by all applications, resulting in code reuse and reduced code size; second, programmers do not have to deal with the implementation of any security mechanism; this significantly simplifies the development of applications, also in presence of secure communications. Unfortunately, as described in the previous sections, both IPSec and (D)TLS have their own drawbacks. Probably the main issue that is common to both IP and transport approaches is due to the impossibility to assure complete end-to-end security when application communications are relayed by intermediate nodes that work at application level (e.g., proxies). In this case end-to-end security can be still provided with transport or IP level mechanisms, but only in the presence of very trusted intermediate systems. However, in this case, the overall security is complicated by the handling of such hop-by-hop trust management.

A different approach aiming at providing complete end-to-end security is to enforce security directly at application level. This of course simplifies the requirements for underlying layers, and probably reduces the cost, in term of packet size and data processing, since only application data have to be secured and per-data and not per-packet overhead is introduced. Moreover, multicast communications, and in-network data aggregation in encrypted domains (for example through homomorphic cryptography) is easier to be implemented at application level.

The main disadvantages of providing security at application level are the complications introduced for application development and the overall code size due to poor reuse of software codes. This is mainly due to the lack of well defined and adopted secure protocols at application level. Examples of standards that can be used for this purpose are S/MIME and SRTP. S/MIME (Secure/Multipurpose Internet Mail Extensions) [20] is a standard for providing authentication, message integrity, non-repudiation of origin, and confidentiality for application data. Although S/MIME has been originally developed for securing MIME data between mail user agents, it is not restricted to mail and can be used for securing any application data and encapsulated within any application and transport protocols. SRTP (Secure Real-time Transport Protocol) [21] is another secure communication protocol that provides confidentiality, message authentication, and replay protection to application data. It is an extension of the Real-time Transport Protocol (RTP) specifically developed for handling real-time data communications (e.g., voice or video communication), but can be re-used also in other application scenarios. It works in a per-packet fashion and is usually encapsulated in UDP. However, more investigation is required to state which is the standard protocol most suitable for securing data at application level in network and node constrained scenarios such as for IoT.

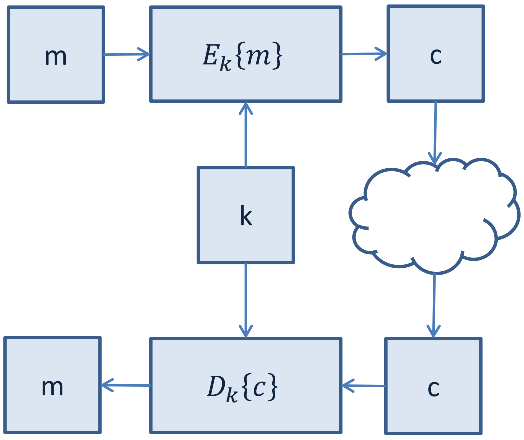
3. Lightweight Cryptography for Smart Objects

The development of the IoT will result in the extensive deployment of billions of smart objects that will interact with the existing Internet. Smart objects are tiny computing devices, featuring constrained resources, such as low computation capabilities, little memory, and limited battery. Communication with smart objects in resource-constrained environments must necessarily take into account these hard limitations, especially in scenarios where security is a crucial aspect and conventional cryptographic primitives, such as the Advanced Encryption Standard (AES) [22], are inadequate.

Lightweight Cryptography (LWC) is a very interesting research area, aiming at the design of new ciphers that might meet the requirements set by the use of smart objects [23]. The term "lightweight" should not be mistaken with weak (in terms of cryptographic protection), but should instead be interpreted as referring to a family of cryptographic algorithms with smaller footprint, low energy consumption, and low computational power needs. These ciphers aim at providing sufficient security in the environment of restricted resources as can be found in many ubiquitous devices [24]. LWC thus represents a cryptography tailored to constrained devices, which must cope with the trade-offs between security level, cost, and performance.

In this section, an overview of the most prominent cryptographic algorithms is presented, followed by a comparison among lightweight cryptographic primitives and conventional ones, such as AES, which are currently adopted by standard Internet security protocol, such as IPSec and TLS. Symmetric ciphers for lightweight cryptography are presented first, followed by asymmetric ciphers. Cryptographic hash functions are discussed next. Finally, privacy homomorphism is discussed. We remark that this overview is not meant to be detailed or extensive, but aims at pointing out which encryption algorithms are most suitable for practical implementation in IoT scenarios.

Figure 3. Secure communication with symmetric-key cryptographic algorithms.



3.1. Symmetric-Key LWC Algorithms

Symmetric-key cryptographic algorithms use the same key for encryption of a plaintext and decryption of a ciphertext. The encryption key represents a shared secret between the parties that are involved in the secure communication. An illustrative representation of symmetric-key secure communication is shown in Figure 3.

Symmetric-key encryption can use either block ciphers or stream ciphers:

- *Block ciphers* operate on fixed-length groups of bits, called blocks, padding the plaintext to make its length equal to a multiple of the block size. An example is the AES algorithm;
- In *stream ciphers* the digits of a plaintext are encrypted one at a time with the corresponding digit of a pseudorandom cipher digit stream (keystream).

3.1.1 Tiny Encryption Algorithm (TEA)

The Tiny Encryption Algorithm (TEA) is a block cipher renowned for its simplicity of description and implementation, typically a few lines of code [25]. TEA operates on two 32-bit unsigned integers (could be derived from a 64-bit data block) and uses a 128-bit key. TEA relies only on arithmetic operations on 32-bit words and uses only addition, XORing, and shifts. TEA uses a large number of iterations, rather than a complicated program, in order to avoid preset tables and long setup times. The main design goal of TEA is to define a simple and short cipher that does not rely on preset tables or pre-computations, thus leading to a smaller footprint.

TEA was later revised in order to fix some weaknesses found in the original algorithm, such as the problem of equivalent keys, which reduced the actual key size from 128 to 126 bits. The redesign of TEA, named XTEA (Extended TEA) [26], fixes this problem by changing the key schedule. XTEA also requires two fewer additions, thus resulting in a slightly faster algorithm. Other modifications of the TEA algorithm have been presented, such as XXTEA, Block TEA, Speed TEA, and Tiny XTEA.

As the TEA family uses exclusively very simple operations (addition, XORing, shifts), and has a very small code size, it is an ideal candidate as a cryptographic algorithm to implement security mechanisms in smart objects and wireless sensors.

3.1.2 Scalable Encryption Algorithm (SEA)

The Scalable Encryption Algorithm (SEA) is targeted for small embedded applications [27]. The design considers a context with very limited processing resources and throughput requirements. Another design principle of SEA is flexibility: the plaintext size n, key size n, and processor (or word) size b are design parameters, with the only constraint that n is a multiple of $6b$; for this reason, the algorithm is denoted as SEA$_{n,b}$. The motivation of this flexibility is the observation that many encryption algorithms perform differently depending on the platform, e.g., 8-bit or 32-bit processors. SEA$_{n,b}$ is designed to be generic and adaptable to different security levels (by varying the key size) and target hardware. A great advantage of SEA$_{n,b}$ is the "on-the-fly" key derivation. The main disadvantage is that SEA$_{n,b}$ trades space for time and this may not be negligible on devices with limited computational power.

3.1.3 PRESENT Cipher

PRESENT is an ultra-lightweight block cipher algorithm based on a Substitution-Permutation Network (SPN) [28]. PRESENT has been designed to be extremely compact and efficient in hardware. It operates on 64-bit blocks and with keys of either 80 or 128 bits. It is intended to be used in situations where low-power consumption and high chip efficiency are desired, thus making it of particular interest for constrained environments. The main design goal of PRESENT is, as for the other lightweight ciphers, simplicity. PRESENT is performed in 31 rounds, each comprising three stages:

- key-mixing, through XOR operation and a 61-bit rotation key schedule;
- substitution layer, through 16 4-bit (input) by 4-bit (output) S-boxes;
- permutation layer.

At the end of the 31st round, an additional round is performed by XORing the last round subkey.

ISO/IEC 29192-2:2012 "Lightweight Cryptography" specifies PRESENT as a block cipher suitable for lightweight cryptography [29].

3.1.4 Hight

The HIGh security and light weigHT (HIGHT) encryption algorithm is a generalized Feistel network with a block size of 64 bits, 128-bit keys and 32 rounds [30]. HIGHT was designed with an eye on low-resource hardware performance. HIGHT uses very simple operations, such as XORing, addition mod 2^8, and bitwise rotation. The key schedule in HIGHT is designed so that subkeys are generated on the fly both in the encryption and the decryption phases.

3.1.5 Comparison of Symmetric LWC Algorithms

LWC algorithms are not intended to supersede existing ciphers, e.g., AES, for widespread use. Their application is limited to those scenarios where classical ciphers might be inefficient, such as scenarios where:

- a moderate security level is required, so that keys need not be too long;
- encryption should not be applied to large amounts of data;
- the hardware area needed for implementation and the power consumption are considered harder requirements than speed.

For constrained devices, the choice of the cryptographic algorithm is a primary element that can affect performance. When low cost and energy consumption are hard requirements, computational power must inherently be downsized accordingly. Using 8-bit microcontrollers (such as Atmel AVR microcontrollers [31]), which have limited capabilities in terms of computing power, memory, and storage, requires that implemented ciphers have small footprint and are kept simple. This may result in faster execution and thus in lower energy consumption, which may be critical for battery-powered devices.

Although most symmetric cryptographic algorithms have been developed focusing on efficient software implementations, the deployment of smart objects will lead to an increasing attention to those ciphers that will perform well in hardware in terms of speed and energy consumption. In Table 1, we report a direct comparison, presented in [23], among the LWC algorithms outlined in Subsection 3.1, with particular reference to the following metrics: key size, block size, rounds, consumed area measured in gate equivalents (GEs), code size (in bytes). Reported values for gate equivalents are related to hardware implementations, while code size refers to software implementations.

Table 1. Comparison of different symmetric-key cryptographic algorithms. Key size and Block size are expressed in number of bits. Code size is expressed in number of bytes.

	Cipher	**Key size**	**Block size**	**Rounds**	**GE** (hardware impl.)	**Code size** (software impl.)
SOFTWARE CIPHERS	AES	128	128	10	3400 [32,33]	2606
	TEA	128	64	32	3490 [34]	1140
	SEA$_{96,8}$	96	8	$\geq 3n/4$	3758 [1] [35] 3925 [2] [35] 2547 [36]	2132
HARDWARE CIPHERS	PRESENT	80	64	32	1570 [28]	936
	HIGHT	128	64	32	3048 [30]	5672

[1] Round-based implementation with datapath of size n; [2] Serialized implementation with datapath of size b.

3.2. Public-Key (Asymmetric) LWC Algorithms

Public-key (asymmetric) cryptography requires the use of a public key and a private key. Public keys can be associated with the identity of a node by including them into a public certificate, signed by a Certification Authority (CA) that can be requested to verify the certificate. Public-key cryptography requires the significant effort of deploying a Public Key Infrastructure (PKI). Moreover, asymmetric cryptography requires higher processing and long keys (at least 1024 bits for RSA [37]) to be used. Alternative public-key cryptographic schemes, such as Elliptic Curve Cryptography (ECC) [38], might require shorter keys to be used in order to achieve the same security than RSA keys. However, because of these reasons, symmetric cryptography is preferred in terms of processing speed, computational effort, and size of transmitted messages. Public-key ciphers are usually used to setup symmetric keys to be used in subsequent communications.

3.2.1 RSA Algorithm

The Rivest, Shamir, and Adleman (RSA) algorithm is the most known and widely used public-key scheme. It is based on exponentiation in a finite field over integers modulo N. Consider a modulus N and a pair of public and private keys (e, d). The encryption of a message m is given by $c = m^e$ mod N, while the decryption is $m = c^d$ mod N. The key generation phase of RSA, aiming to generate the public-private key pair, consists of the following steps:

1. select two large prime numbers denoted as p and q such that $p \neq q$;
2. compute $n = p \cdot q$;

3. compute the Euler's totient function $\Phi(n) = (p-1) \cdot (q-1)$;
4. choose an integer e such that $1 < e < \Phi(n)$ and that the $GCD(e,\Phi(n))$ = 1;
5. compute $d = e^{-1}$ mod $\Phi(n)$;
6. the pair (n, e) is the public key, while d is the private key.

The security of the RSA algorithm depends on the hard problem of factorizing large integers. In order to achieve an acceptable level of security, n should be at least 1024 bits long, so that p and q and consequently $\Phi(n)$ cannot be obtained, thus protecting the (e, d) pair.

RSA is unsuitable for adoption in constrained devices due to the need to operate on large numbers and the fact that long keys are required to achieve sufficient security. Moreover, both key generation and encryption/decryption are demanding procedures that result in higher energy consumption.

3.2.2 Elliptic Curve Cryptography (ECC)

ECC is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. While RSA is based on exponentiation on finite fields, ECC depends on point multiplication on elliptic curves. An elliptic curve E over the finite field K (whose characteristic is not equal to 2 and 3) is defined as:

$$E(K) : y^2 = x^3 + ax + b \text{ with } a, b \in K$$

Points $P = (x, y) \in E(K)$ form an Abelian group, thus point addition and scalar point multiplication can be performed.

ECC provides higher security and a better performance than the first generation public-key techniques (RSA and Diffie–Hellman). Moreover, ECC is the most interesting public-key cryptographic family for embedded environments because it can reach the same security level as RSA with much shorter keys, as shown in Table 2 (http://www.nsa.gov/business/programs/elliptic_curve.shtml), and with computationally lighter operations, like addition and multiplication, rather than exponentiation.

Table 2. Comparison of security levels between symmetric ciphers, ECC, and RSA (Recommended NIST key sizes).

Symmetric-key size (bits)	80	112	128	192	256
ECC Key size (bits)	160	224	256	384	512
RSA Key size (bits)	1024	2048	3072	7680	15360

ECC has been accepted commercially and has also been adopted by standardizing institutions such as the American National Standards Institute (ANSI), the Institute of Electrical and Electronics Engineers (IEEE), the International Organization for Standardization (ISO), the Standards for Efficient Cryptography Group (SECG), and the National Institute of Standards and Technology (NIST) [39–43].

The implementation of a lightweight hardware ECC processor for constrained devices is attracting a growing interest. In [23], a possible hardware implementation of a low-area, standalone, public-key engine for Elliptic Curve Cryptography, with a 113-bit binary field for short-term security and a 193-bit binary field for medium-term security, is presented. The choice of use of a binary field, rather than a prime field, is related to the corresponding carry-free arithmetic, which fits well in hardware implementations. With respect to other ECC hardware implementations, the one presented uses a smaller area (in terms of GEs) and faster execution.

3.2.3 Performance Comparison of Available Implementations of Public-key Cryptographic Algorithms

Hereafter, we recall the performance results, presented in [44], of RSA and ECC public-key different available implementations, with benchmarks obtained in constrained devices (namely an 8-bit Arduino Uno board), such as TinyECC and Wiselib. Table 3 shows implementation results for RSA public-key encryption, when the private key is held in SRAM or in ROM. In Table 4, a performance comparison in the ECDSA signature algorithms, using TinyECC and Wiselib implementations, is presented. Finally, a ROM footprint comparison is shown in Table 5.

Table 3. RSA private key operation performance.

Key length (bits)	Execution time (ms)		Memory footprint (bytes)	
	Key in SRAM	Key in ROM	Key in SRAM	Key in ROM
64	66	70	40	32
128	124	459	80	64
512	25089	27348	320	256
1024	109666	218367	640	512
2048	1587559	1740267	1280	104

Table 4. ECDSA signature performance: TinyECC versus Wiselib implementations.

Curve parameters	Execution time (ms)		Memory footprint (bytes)		Comparable RSA key length
	TinyECC	Wiselib	TinyECC	Wiselib	
128r1	1858	10774	776	732	704
128r2	2002	10615	776	732	704
160k1	2228	20164	892	842	1024
160r1	2250	20231	892	842	1024
160r2	2467	20231	892	842	1024
192k1	3425	34486	1008	952	1536
192r1	3578	34558	1008	952	1536

Table 5. Public-key encryption library ROM occupancy.

Library	AvrCryptolib	Wiselib	TinyECC	Relic-toolkit
ROM Footprint (Kbytes)	3.6	16	18	29

3.3. Lightweight Cryptographic Hash Functions

Cryptographic Hash Functions, such as MD5 [45] and SHA-1 [46], are an essential part for any protocol that uses cryptography. Hash functions are used normally for different purposes, such as message integrity check, digital signatures, and fingerprinting. Cryptographic hash functions should ideally be:

- computationally inexpensive;
- pre-image resistant: given a hash h, it should be difficult to invert the hash function in order to obtain the message m such that $h = hash(m)$;
- second pre-image resistant: given a message m_1, it should be difficult to find another message m_2 such that $hash(m_1) = hash(m_2)$;
- collision resistant: it should be difficult to find two messages m_1 and m_2, with $m_1 \neq m_2$, such that $hash(m_1) = hash(m_2)$ (hash collision).

In general, for a hash function with an n-bit output, pre-image and second pre-image resistance require 2^n operations, while collision resistance requires $2^{n/2}$ operations [47]. While the design of standard cryptographic hash functions does not focus on hardware efficiency, lightweight cryptographic hash functions are needed for use in resource-constrained devices in order to minimize the amount of hardware (in terms of GEs) and energy consumption.

In this subsection, we will overview some proposals for lightweight cryptographic hash functions that go beyond classical MD and SHA families.

3.3.1 DM-PRESENT and H-PRESENT

The authors in [47] propose DM-PRESENT and H-PRESENT, two lightweight hash functions based on the PRESENT block cipher. DM-PRESENT is a 64-bit hash function and comes in two versions: DM-PRESENT-80 and DM-PRESENT-128, depending on which cipher (PRESENT-80 or PRESENT-128) is used as a basis. H-PRESENT (namely H-PRESENT-128) is a 128-bit hash function based on the PRESENT-128 block cipher. In their work, the authors also consider the problem of constructing longer hash functions based on the PRESENT block cipher in order to improve the security level.

3.3.2 PHOTON

PHOTON [48] is a hardware-oriented family of cryptographic hash functions designed for constrained devices. PHOTON uses a sponge-like construction [49] as domain extension algorithm and an AES-like primitive as internal unkeyed permutation. A PHOTON instance is defined by its output size ($64 \leq n \leq 256$), its input rate r and its ouptut rate r' (PHOTON-$n/r/r'$). The use of sponge functions framework aims at keeping the internal memory usage low. The framework has been extended in order to increase speed when hashing small messages, which is typically inefficient with sponge functions framework.

3.3.3 SPONGENT

SPONGENT [50] is a family of lightweight hash functions with output of 88, 128, 160, 224, and 256 bits. SPONGENT is based on a sponge construction with a PRESENT-type permutation. An instance of SPONGENT is defined by the output size n, the *rate* r, and the *capacity* c (SPONGENT-$n/c/r$). The size of the internal state, denoted as *width*, is $b = r + c \geq n$. The implementations in an ASIC hardware require 738, 1060, 1329, 1728, and 1950 GEs, respectively, making it the hash function with the smallest footprint in hardware. The 88-bit hash size is used only to achieve pre-image resistance.

3.3.4 QUARK

The QUARK [51] hash family comes with three instances: U-QUARK, D-QUARK, and S-QUARK, with hash sizes of 136, 176, and 256 bits, respectively. QUARK, like PHOTON and SPONGENT, is based on a sponge construction. The QUARK hash family is optimized for hardware implementation and, as stated by the authors, software implementations should instead rely on other designs. QUARK has a bigger footprint than PHOTON and SPONGENT, but shows higher throughput than SPONGENT and higher security than PHOTON.

3.3.5 KECCAK

KECCAK [52] is a family of sponge functions. KECCAK uses a sponge construction in which message blocks are XORed into the initial bits of the state, which is then invertibly permuted. In the version used in KECCAK, the state consists of a 5×5 array of 64-bit words, 1600 bits total. KECCAK produces arbitrary output length.

KECCAK has been selected by the NIST as the winner of the SHA-3 competition [53] on October 2, 2012 and is now referenced as SHA-3.

3.3.6 SQUASH

SQUASH (SQUare-hASH) [54] is suited to challenge-response MAC applications in constrained devices, such as RFID tags. SQUASH is completely deterministic, so it requires no internal source of randomness. SQUASH offers a 64-bit pre-image resistance. SQUASH is not collision resistant, but this is not an issue since it targets RFID authentication protocols, where collision resistance is not needed. However, if collision resistance is a requirement, for instance in case of digital signatures, SQUASH is unsuitable and other hash functions should be considered.

3.4. Homomorphic Encryption Schemes

Homomorphic encryption is a form of encryption that allows specific types of computations to be executed on ciphertexts and obtain an encrypted result that is the ciphertext of the result of operations performed on the plaintext. By denoting $E\{\cdot\}$ as the homomorphic encryption function and $f(\cdot)$ as the computation function, it holds that:

$$E\{f(a, b)\} = f(E\{a\}, E\{b\})$$

An example of homomorphic encryption is the RSA algorithm. Consider a modulus N and an exponent e. The encryption of a message m is given by $E\{m\} = m^e \bmod N$. The homomorphic property holds, since:

$$E\{m_1 \cdot m_2\} = (m_1 \cdot m_2)^e \bmod N = (m_1)^e \bmod N \cdot (m_2)^e \bmod N = E\{m_1\} \cdot E\{m_2\}$$

Other examples of homomorphic encryption schemes are the ECC encryption [38], the ElGamal cryptosystem [55] and the Pailler cryptosystem [56].

Homomorphic encryption is receiving a growing interest for application into IoT scenarios, since it could be used to preserve confidentiality among the endpoints of communication, while making it possible for intermediate nodes to be able to process the information without the need to decrypt the data prior to processing. Homomorphic cryptosystems usually require higher computation and need longer keys to achieve a comparable security level than symmetric-key algorithms.

Depending on the operation $f(\cdot)$ that can be performed on the encrypted data, the homomorphic encryption scheme can be defined as *additive* or *multiplicative*. Additive homomorphism makes it possible to compute sums, subtractions, and scalar multiplication of its operands; multiplicative homomorphism allows to compute the product of its operands. The RSA algorithm is an example of multiplicative homomorphic encryption. An example of additive homomorphic encryption is the Pailler cryptosystem. Given a modulus n, a shared random integer g, and user-generated random integers r_1 and r_2, the homomorphic property is:

$$E\{m_1\} \cdot E\{m_2\} = (g^{m_1} r_1^n \bmod n^2) \cdot (g^{m_2} r_2^n \bmod n^2) = (g^{m_1+m_2})(r_1 r_2)^n \bmod n^2 = E\{m_1 + m_2\}$$

Homomorphic encryption schemes that are either additive or multiplicative are denoted as *partially homomorphic*. If both addition and multiplication are supported, a cryptosystem is denoted as *fully homomorphic*. Fully homomorphic cryptosystems preserve the ring structure of the plaintexts and, therefore, enable more complex procedures. The investigation for fully homomorphic encryption schemes is still at early stages and no practical scheme has been found with acceptable performance (e.g., in terms of decryption delay). Application to IoT scenarios is a rich research topic.

4. Key Agreement, Distribution, and Security Bootstrapping

Key distribution and management is a crucial issue that needs to be addressed when security mechanisms have to be adopted. Key agreement protocols have been around for years: the Diffie–Hellman key exchange protocol is an example of a key agreement protocol that two parties perform in order to setup a shared key to be used in a session [57]. Other mechanisms have been defined and implemented. The Internet Key Exchange (IKE) [58] protocol is a the protocol defined to setup a Secure Association to be used in IPSec.

4.1. Key Agreement Protocols

Asymmetric (public-key) cryptographic algorithms are often the basis for key agreement protocols, yet other techniques that do not involve the adoption of asymmetric cryptography have been proposed. A polynomial-based key pre-distribution protocol has been defined [59] and applied to Wireless Sensor

Networks in [60]. A possible alternative key agreement protocol is SPINS [61], which is a security architecture specifically designed for sensor networks. In SPINS, each sensor node shares a secret key with a base station, which is used as a trusted third-party to set up a new key, with no need of public-key cryptography. The authors of [62] present three efficient random key pre-distribution schemes for solving the security bootstrapping problem in resource-constrained sensor networks, each of which represents a different tradeoff in the design space of random key protocols.

4.2. Shared Group-Key Distribution

The above mechanisms apply to scenarios where communication occurs between two parties (unicast and point-to-point communications). In different communication scenarios, such as point-to-multipoint (multicast) or multipoint-to-point communications, other mechanisms must be investigated and adopted. In such scenarios, the adoption of a shared group-key is appealing.

Secure group communications provide confidentiality, authenticity, and integrity of messages exchanged within the group, through the use of suitable cryptographic services and without interfering with the communication data path. In order to achieve secure group communications, nodes must share some cryptographic material that must be handled properly so that group membership changes, both predictable and unpredictable, can be managed. In fact, any membership change should trigger a *rekeying* operation, which is intended to update and redistribute the cryptographic material to the group members, so that: (i) a former member cannot access current communication (*forward secrecy* [63]); (ii) a new member cannot access previous communication (*backward secrecy* [64]). The authors in [65] define a DTLS record based approach to secure multicast communication in lossy low-power networks.

Assuming that the used cryptographic primitives cannot be broken by an attacker with limited computational power (*i.e.*, for whom it is infeasible to carry out a brute force attack in order to solve the problems behind cryptographic schemes, such as discrete logarithm, or inverting MD5/SHA-1), the main challenge is the distribution of the group keys and their updates: this problem is referred to as *Group Key Distribution* (GKD). The GKD problem can be tackled according to the two different approaches:

- current communications can be deciphered independently of previous communications (stateless receivers): this approach is denoted as *broadcast encryption* [66,67];
- users maintain state of the past cryptographic material (stateful receivers): this approach is termed *multicast key distribution* [68].

In the case of a multicast key distribution, centralized [69] or distributed [70] approaches can be adopted. In a distributed approach, the group key is computed and maintained by the group members themselves: an example of a distributed approach is the Tree-based Group Diffie–Hellman (TGDH) protocol [71]. In a centralized approach, the task of key distribution is assigned to a single entity, denoted as Key Distribution Center (KDC). This approach allows to have a simple mechanism with a minimal number of exchanged messages. Logical Key Hierarchy (LKH) [63] and MARKS [72] are key distribution protocols that try to optimize the number of exchanged messages between a KDC and the group members. LKH is based on key graphs, where keys are arranged into a hierarchy and the KDC maintains all the keys. MARKS is a scalable approach and does not need any update message when members join or leave the group predictably. However, MARKS does not address the issue of member eviction with a subsequent key revocation.

4.3. Security Bootstrapping

All key agreement protocols require that some credentials, either public/private key pairs, symmetric keys, certificates, or others, have been previously installed and configured on nodes, so that the key agreement procedure can occur securely. Bootstrapping refers to the processing operations required before the network can operate: this requires that proper setup, ranging from link layer to application layer information, must be done on the nodes. Bootstrapping is a very important phase in the lifecycle of smart objects and can affect the way they behave in operational conditions. Even though the bootstrapping phase is outside the scope of this paper, it is important to consider security bootstrapping mechanisms and architecture [73], so that possible threats, such as cloning or malicious substitution of objects, can be tackled properly. The author of [74] provides a sketch of a possible protocol to let constrained devices to securely bootstrap into a system that uses them.

5. Processing Data in the Encrypted Domain: Secure Data Aggregation

In-network data aggregation in wireless sensor networks consists in executing certain operations (such as sums and averages) at intermediate nodes in order to minimize the amount of transmitted messages and the processing load at intermediate nodes, so that only significant information is passed along in the network. This leads to several benefits, such as energy savings, which are crucial for constrained environments, such as low-power and lossy networks. Data aggregation refers to a multipoint-to-point communication scenario, which requires intermediate nodes to operate on received data and forward the output of a suitable function applied to such input data. In those scenarios, where privacy on transmitted data is an issue, it might be required to send encrypted data. Encryption can be adopted not only to achieve confidentiality, but also to verify the authenticity and integrity of messages.

While secure data aggregation is certainly an application-related aspect also in WSNs, optimized communication can have a positive impact also in some IoT scenarios. Smart parking or critical infrastructures scenarios could benefit from minimizing transmitted data, possibly by adopting privacy homomorphism algorithms, discussed in Section 3.

A simple aggregation strategy can be to queue the payloads of the received packets and send out only one packet with all the information. This approach can bring only limited gains, since only the payloads are considered. Another, more efficient, approach can be used if the aggregator is aware of the type of operation that the final recipient is willing to perform. Consider a scenario where a node is interested in counting all the nodes in the network. Nodes send a packet with “1” as a content. Aggregators receive these packets and can just sum the “1”s received and send out one packet with the same size whose content is the number of “1”s received. By doing this, the information sent across the network is minimal and the final recipient must perform simpler processing.

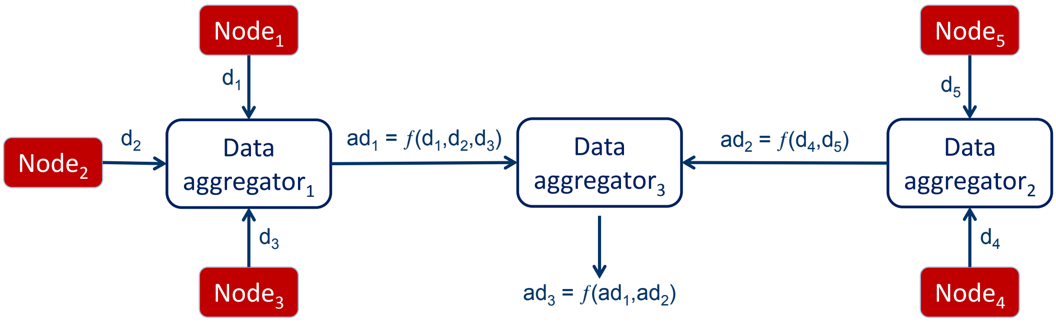
Typically, secure data aggregation mechanisms require nodes to perform the following operations:

1. at the transmitting node, prior to transmission, data are encrypted with some cryptographic function E;

2. at the receiving node, all received data packets are decrypted with the inverse cryptographic function $D = E^{-1}$ to retrieve the original data;
3. data are aggregated with an aggregation function;
4. prior to retransmission, aggregated data are encrypted through E and relayed to the next hop.

This process is iterated at intermediate nodes until the data reach the destination node that is interested in receiving the result of aggregation, as shown in Figure 4. Both symmetric and asymmetric cryptographic schemes can be applied.

Figure 4. In-network data aggregation.



The aggregation procedure just explained raises the following issues, especially if we consider a scenario where the aggregators are not special nodes, but have the same features as other nodes in the network.

1. Aggregators must decrypt each incoming piece of information before processing, in order to perform the aggregation and, subsequently, encrypt the result before transmission to the next hop. This has clearly an impact on the computation and, therefore, on the energy consumption of the aggregator.
2. An aggregator must keep a secure association (*i.e.*, share a symmetric key) with any node that either sends data to or receives data from it. This further introduces the need for increased complexity at the aggregator.
3. Intermediate aggregators access the data that they receive, even though they are not intended to do so, since the actual recipient of the data is another node. This might introduce privacy concerns, especially in scenarios where intermediate nodes might not have a trust relationship with the sender.

In order to cope with the problems sketched above, some actions can be considered. All these issues can be addressed by using homomorphic encryption schemes, introduced in Section 3.4. Homomorphic encryption can be used to avoid the need to decrypt the information that must be aggregated and encrypt the result and operate on the encrypted data directly. This can dramatically increase the performance, in terms of execution time and energy savings, since the encryption/decryption operations are typically computationally demanding. Besides computational and energy efficiencies, a positive side effect of the adoption of homomorphic encryption is the fact that only the sources and the final destination of the data are capable of accessing the real data, thus preserving "end-to-end" confidentiality for the aggregation application scenario. Additional security can be introduced by using *probabilistic* cryptosystems, such as the Pailler cryptosystem: in this case, given two encrypted values, it is not possible to decide whether they conceal the same value or not. This is especially useful to avoid that eavesdroppers can determine the content of a secure communication just by observing the encrypted packets.

6. Authorization Mechanisms for Secure IoT Services

Authorization mechanisms should be considered when deploying IoT services, in order to tackle possible concerns that the deployment of smart objects and services relying on them might raise in public opinion. In particular, authorization mechanisms must address the following questions:

- Which users can access some given data?
- How should the information be presented to a given user?
- Which operations is a user allowed to perform?

Role-based Access Control (RBAC) and Attribute-based Access Control (ABAC) are the most widespread approaches to restricting system access to authorized users. RBAC maps permissions to roles that a user has been assigned. On the other hand, ABAC maps permissions to attributes of the user. However, authorization mechanisms strongly depend on an authentication step that must have been previously taken, in order to identify users. An example of a complex, context-aware Access Control system designed for a Medical Sensor Networks scenario, which has critical privacy and confidentiality issues, is described in [75].

Just like popular Internet-based services, such as social networks, have already faced the urge to solve privacy-related problems when dealing with personal and protected data that might be made accessible to third-parties, IoT applications are also going to be facing the same issues. Since IoT services are expected to be offered in a RESTful paradigm, like those cited above, it could be helpful to borrow ideas from the experience that has been already created with Internet REST services.

The OAuth (Open Authorization) protocol has been defined to solve the problem of allowing authorized third-parties to access personal user data [76]. The OAuth protocol defines the following three roles.

- *Resource Owner*: an entity capable of granting access to a protected resource, such as an end-user.
- *Resource Server* (Service Provider, SP): a server hosting user-related information.
- *Client* (Service Consumer, SC): a third-party willing to access personal user's data to reach its goals.

An additional role is defined: an *Authorization Server* (AS), which issues access tokens to the client after successfully authenticating the resource owner and obtaining a proper authorization.

In a general scenario, a SC, which has been previously authorized by a user, can access the data that the user has made visible to the SC. This can be achieved by letting the SC retrieve the data from the SP on the user's behalf. In order to do so, one possible approach could be to force the user to give out personal authentication credentials to the SC. This approach has many drawbacks:

- the SC is going to appear to the SP just like the actual user, thus having unlimited access to the user's personal data;

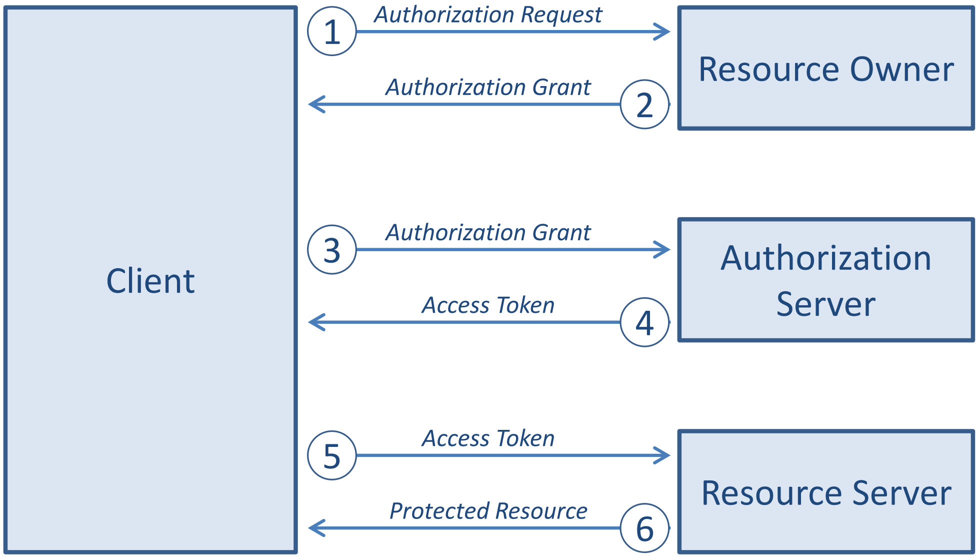
- the user cannot define different restrictions for different SC;
- the user cannot revoke the grant to a SC, unless it changes its credentials.

It is thus necessary to provide a mechanism that can separate the different roles. This can be done by granting specific credentials that the SC can exhibit to the SP, which contain information about its identity and the user's identity, so that the SP can serve its requests according to the access policies that the user has defined for the SC. The OAuth protocol defines the mechanisms that are needed to grant, use, and verify these credentials, named "access tokens".

The OAuth protocol defines the following flow of interaction between the four roles introduced above, as illustrated in Figure 5.

1. The client requests authorization from the resource owner: the authorization request can be made directly to the resource owner or, preferably, indirectly via the AS as an intermediary.
2. The client receives an authorization grant, which is a credential representing the resource owner's authorization.
3. The client requests an access token by authenticating with the AS and presenting the authorization grant.
4. The authorization server authenticates the client, validates the authorization grant, and if the grant is valid, issues an access token.
5. The client requests the protected resource from the resource server and authenticates by presenting the access token.
6. The SP validates the access token and, if valid, serves the request.

Figure 5. Interaction between the four roles of the OAuth protocol flow.



The OAuth authorization framework (currently in version 2.0 [77]) enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.

For IoT scenarios, when using CoAP as an application-layer protocol instead of HTTP, a modified version of OAuth should be defined in order to ensure an authorization layer for restricting access to smart objects services. Since OAuth is an open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications, it has to be adapted in order to fit into IoT application scenarios and to be compatible with constrained environments. For instance, header compression should be used to minimize the amount of information sent along.

HTTP/CoAP proxies should be able to perform OAuth proxying as well, in order to allow interoperability between conventional and constrained OAuth clients and servers. This raises particular challenges since the OAuth specification recommends the usage of HTTPS as a means to avoid man-in-the-middle (MITM) attacks, thus preventing the access tokens to be stolen and used by malicious nodes. This means that CoAPs should be used in order to comply with the specification. The HTTP/CoAP proxy should then be able to perform a TLS-to-DTLS mapping in order to ensure end-to-end security. However, the use of HTTPS (and CoAPs inherently) can be avoided: it is possible to use OAuth over an insecure communication channel by adopting digital signature schemes with HMAC-SHA1 and RSA-SHA1.

7. Conclusions

In this work, security in emerging IoT scenarios was analyzed from a multi-layer perspective. The security protocols of the original Internet stack have been discussed together with the challenges that their use in constrained IoT environments might raise. Compression and optimization of the existing Internet protocols, including security protocols, has also been discussed.

Together with security protocols, another important aspect related to security is the definition and implementation of cryptographic algorithms, tailored for constrained devices. Lightweight cryptographic algorithms have been overviewed and compared in order to highlight the main features of each and to isolate those that are more suitable for use in constrained environments.

Key agreement protocols and group key distribution mechanisms for secure group communication have been discussed, since it may be particularly relevant in scenarios where several smart objects need to communicate and cooperate in order to perform some common task. Related to this aspect, secure data aggregation schemes through homomorphic encryption algorithms have been also discussed, as a possible solution to minimize the processing of smart objects and optimize data transmission.

Finally, the problem of service authorization has been presented. Authorization represents a highly relevant security property that must be dealt with in deploying services in smart objects, since these may raise concerns about privacy and the use of personal data. The future of the IoT will have to cope with this issue, which spans from an ethical perspective to technical aspects.

Acknowledgements

This work is funded by the European Community's Seventh Framework Programme, area "Internetconnected Objects", under Grant no. 288879, CALIPSO project-Connect All IP-based Smart Objects. The work reflects only the authors' views; the European Community is not liable for any use that may be made of the information contained herein.

References

1. Deering, S.; Hinden, R. RFC 2460-Internet Protocol, Version 6 (IPv6) Specification, 1998. Available online: http://tools.ietf.org/rfc/rfc2460.txt (accessed on 31 January 2013).
2. Heer, T.; Garcia-Morchon, O.; Hummen, R.; Keoh, S.L.; Kumar, S.S.; Wehrle, K. Security challenges in the IP-based internet of things. *Wirel. Pers. Commun.* **2011**, *61*, 527–542.
3. Garcia-Morchon, O.; Keoh, S.; Kumar, S.; Hummen, R.; Struik, R. *Security Considerations in the IP-based Internet of Things*; IETF Internet Draft draft-garcia-core-security-04; The Internet Engineering Task Force (IETF): Fremont, CA, USA, 2012.
4. Bormann, C. *Guidance for Light-Weight Implementations of the Internet Protocol Suite*; IETF Internet Draft draft-ietf-lwig-guidance-02, The Internet Engineering Task Force (IETF): Fremont, CA, USA, 2012.
5. IETF Constrained RESTful Environments Working Group. Available online: http://tools.ietf.org/wg/core/ (accessed on 31 January 2013).
6. IETF IPv6 over Low power WPAN. Available online: http://tools.ietf.org/wg/6lowpan/ (accessed on 31 January 2013).
7. IETF Routing Over Low power and Lossy networks Working Group. Available online: http://tools.ietf.org/wg/roll/ (accessed on 31 January 2013).
8. IETF Light-Weight Implementation Guidance. Available online: http://tools.ietf.org/wg/lwig/ (accessed on 31 January 2013).
9. Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T. RFC 2616, Hypertext Transfer Protocol – HTTP/1.1. 1999. Available online: http://tools.ietf.org/rfc/rfc2616.txt (accessed on 31 January 2013).
10. Shelby, Z.; Hartke, K.; Bormann, C.; Frank, B. *Constrained Application Protocol (CoAP)*; IETF Internet Draft draft-ietf-core-coap-13; The Internet Engineering Task Force (IETF): Fremont, CA, USA, 2012.
11. Kent, S.; Atkinson, R. RFC 2401—Security Architecture for the Internet Protocol, 1998. Available online: http://tools.ietf.org/rfc/rfc2401.txt (accessed on 31 January 2013).
12. Raza, S.; Duquennoy, S.; Chung, T.; Yazar, D.; Voigt, T.; Roedig, U. Securing Communication in 6LoWPAN with Compressed IPsec. In Proceedings of the International Conference on Distributed Computing in Sensor Systems (IEEE DCOSS 2011), Barcelona, Spain, 27–29 June 2011.
13. Moskowitz, R.; Heer, T.; Jokela, P.; Henderson, T. *Host Identity Protocol Version 2 (HIPv2)*; Technical report, IETF Internet Draft draft-ietf-hip-rfc5201-bis-10; The Internet Engineering Task Force (IETF): Fremont, CA, USA, 2012.
14. Jokela, P.; Moskowitz, R.; Melen, J. *Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP)*; IETF Internet Draft draft-ietf-hip-rfc5202-bis-01; The Internet Engineering Task Force (IETF): Fremont, CA, USA, 2012.
15. Dierks, T.; Allen, C. RFC 5246-The TLS Protocol, 2008. Available online: http://tools.ietf.org/rfc/rfc5246.txt (accessed on 31 January 2013).
16. Rescorla, E.; Modadugu, N. RFC 6347-Datagram Transport Layer Security Version 1.2, 2012. Available online: http://tools.ietf.org/rfc/rfc6347.txt (accessed on 31 January 2013).

17. Raza, S.; Trabalza, D.; Voigt, T. 6LoWPAN Compressed DTLS for CoAP. In Proceedings of the 8th IEEE International Conference on Distributed Computing in Sensor Systems (IEEE DCOSS 2012), Hangzhou, China, 16–18 May 2012.
18. Brachmann, M.; Keoh, S.; Morchon, O.; Kumar, S. End-to-End Transport Security in the IP-Based Internet of Things. In Proceedings of the Computer Communications and Networks (ICCCN), 2012 21st International Conference on, Munich, Germany, 30 July–2 August 2012; pp. 1–5.
19. Brachmann, M.; Morchon, O.; Keoh, S.; Kumar, S. Security Considerations around End-to-End Security in the IP-Based Internet of Things. In Proceedings of the Workshop on Smart Object Security, in Conjunction with IETF83, Paris, France, 25–30 March 2012.
20. Ramsdell, B.; Turner, S. RFC 3711-Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification, 2010. Available online: http://tools.ietf.org/rfc/rfc5751.txt (accessed on 31 January 2013).
21. Baugher, M.; McGrew, D.; Naslund, M.; Carrara, E.; Norrman, K. RFC 3711-The Secure Real-time Transport Protocol (SRTP), 2004. Available online: http://tools.ietf.org/rfc/rfc3711.txt (accessed on 31 January 2013).
22. Daemen, J.; Rijmen, V. *The Design of Rijndael*; Springer-Verlag New York, Inc.: Secaucus, NJ, USA, 2002.
23. Eisenbarth, T.; Kumar, S.; Paar, C.; Poschmann, A.; Uhsadel, L. A survey of lightweight-cryptography implementations. *IEEE Des. Test* **2007**, *24*, 522–533.
24. Rinne, S.; Eisenbarth, T.; Paar, C. Performance Analysis of Contemporary Light-Weight Block Ciphers on 8-bit. Avaialble online: http://www.emsec.rub.de/research/publications/performance-analysis-contemporary-light-weight-blo/ (accessed on 1 April 2013).
25. Wheeler, D.; Needham, R. *TEA, a Tiny Encryption Algorithm*; Springer-Verlag: New York, NY, USA, 1995; pp. 97–110.
26. Needham, R.M.; Wheeler, D.J. *TEA Extensions*; Technical report; University of Cambridge, Cambridge, United Kingdom, 1997.
27. Standaert, F.X.; Piret, G.; Gershenfeld, N.; Quisquater, J.J. SEA: A Scalable Encryption Algorithm for Small Embedded Applications. In Proceedings of the Smart Card Research and Applications, Proceedings of Cardis 2006, LNCS, Tarragona, Spain, 19–21 April 2006; Springer-Verlag: New York, NY, USA, 2006; pp. 222–236.
28. Bogdanov, A.; Knudsen, L.R.; Le, G.; Paar, C.; Poschmann, A.; Robshaw, M.J.B.; Seurin, Y.; Vikkelsoe, C. PRESENT: An Ultra-Lightweight Block Cipher. In Proceedings of the CHES 2007, Vienna, Austria, 10–13 September 2007; Springer: Berlin/Heidelberg, Germany, 2007.
29. ISO/IEC 29192-2:2012. *Information Technology–Security Techniques–Lightweight Cryptography–Part 2: Block Ciphers*; ISO: Geneva, Switzerland, 2012.
30. Hong, D.; Sung, J.; Hong, S.; Lim, J.; Lee, S.; Koo, B.; Lee, C.; Chang, D.; Lee, J.; Jeong, K.; *et al.* HIGHT: A New Block Cipher Suitable for Low-Resource Device. In Proceedings of the Cryptographic Hardware and Embedded Systems-CHES 2006, 8th International Workshop, Yokohama, Japan, 10–13 October 2006; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4249, *Lecture Notes in Computer Science*, pp. 46–59.

31. Atmel AVR 8-bit Microcontrollers. Available online: http://www.atmel.it/products/microcontrollers/avr/default.aspx (accessed on 31 January 2013).
32. Feldhofer, M.; Dominikus, S.; Wolkerstorfer, J. Strong Authentication for RFID Systems Using the AES Algorithm. In *Cryptographic Hardware and Embedded Systems-CHES 2004*; Joye, M., Quisquater, J.J., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; Volume 3156, pp. 357–370.
33. Feldhofer, M.; Wolkerstorfer, J.; Rijmen, V. AES implementation on a grain of sand. *Inf. Security IEEE Proc.* **2005**, *152*, 13–20.
34. Kaps, J.P. Chai-Tea, Cryptographic Hardware Implementations of xTEA. In Proceedings of the 9th International Conference on Cryptology in India: Progress in Cryptology, Kharagpur, India, 14–17 December 2008; Springer-Verlag: Berlin/Heidelberg, Germany, 2008; pp. 363–375.
35. Mac, F.; St, F.; Quisquater, J. ASIC Implementations of the Block Cipher SEA for Constrained Applications. Available online: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.88.926 (accessed on 1 April 2013).
36. Plos, T.; Dobraunig, C.; Hofinger, M.; Oprisnik, A.; Wiesmeier, C.; Wiesmeier, J. Compact Hardware Implementations of the Block Ciphers mCrypton, NOEKEON, and SEA. In *Progress in Cryptology-INDOCRYPT 2012*; Galbraith, S., Nandi, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7668, pp. 358–377.
37. Rivest, R.; Shamir, A.; Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **1978**, *21*, 120–126.
38. Koblitz, N. Elliptic curve cryptosystems. *Math. Comput.* **1987**, *48*, 203–209.
39. American National Standards Institute. Available online: http://www.ansi.org (accessed on 31 January 2013).
40. Institute of Electrical and Electronics Engineers. Available online: http://www.ieee.org (accessed on 31 January 2013).
41. International Organization for Standardization. Available online: http://www.ieee.org (accessed on 31 January 2013).
42. Standards for Efficient Cryptography Group. Available online: http://secs.org (accessed on 31 January 2013).
43. National Institute of Standards and Technology. Available online: http://www.nist.gov (accessed on 31 January 2013).
44. Sethi, M.; Arkko, A.; Keranen, A.; Rissanen, H. *Practical Considerations and Implementation Experiences in Securing Smart Object Networks*; IETF Internet Draft draft-aks-crypto-sensors-02; The Internet Engineering Task Force (IETF): Fremont, CA, USA, 2012.
45. Rivest, R. RFC 1321: The MD5 message-digest algorithm. The Internet Engineering Task Force (IETF): Fremont, CA, USA, 1992.
46. Eastlake, D.E.; Jones, P.E. US Secure Hash Algorithm 1 (SHA1). Available online: http://www.ietf.org/rfc/rfc3174.txt (accessed on 31 January 2013).
47. Bogdanov, A.; Leander, G.; Paar, C.; Poschmann, A.; Robshaw, M.J.; Seurin, Y. Hash Functions and RFID Tags: Mind the Gap. In Proceedings of the 10th International Workshop on Cryptographic Hardware and Embedded Systems, Washington, DC, USA, 10–13 August 2008; Springer-Verlag: Berlin/Heidelberg, Germany, 2008; CHES ’08, pp. 283–299.

48. Guo, J.; Peyrin, T.; Poschmann, A. The PHOTON Family of Lightweight Hash Functions. In Proceedings of the 31st Annual Conference on Advances in Cryptology, Santa Barbara, CA, USA, 14-18 August 2011; Springer-Verlag: Berlin/Heidelberg, Germany, 2011; CRYPTO'11, pp. 222–239.
49. Bertoni, G.; Daemen, J.; Peeters, M.; Assche, G.V. Sponge functions, Ecrypt Hash Workshop, 24–25 May 2007, Barcelona, Spain.
50. Bogdanov, A.; Knežević, M.; Leander, G.; Toz, D.; Varici, K.; Verbauwhede, I. SPONGENT: A Lightweight Hash Function. In Proceedings of the 13th International Conference on Cryptographic Hardware and Embedded Systems, Nara, Japan, 28 September–1 October 2011; Springer-Verlag: Berlin/Heidelberg, Germany, 2011; CHES'11, pp. 312–325.
51. Aumasson, J.P.; Henzen, L.; Meier, W.; Naya-Plasencia, M. QUARK: A Lightweight Hash. In Proceedings of the 12th International Conference on Cryptographic Hardware and Embedded Systems, Santa Barbara, CA, USA, 17–20 August 2010; Springer-Verlag: Berlin/Heidelberg, Germany, 2010; CHES'10, pp. 1–15.
52. Bertoni, G.; Daemen, J.; Peeters, M.; Assche, G.V. Keccak specifications. Available online: http://keccak.noekeon.org/Keccak-specifications.pdf (accessed on 1 April 2013).
53. NIST SHA-3 Competition. Available online: http://csrc.nist.gov/groups/ST/hash/sha-3 (accessed on 31 January 2013).
54. Shamir, A. *Fast Software Encryption*; Springer-Verlag: Berlin/Heidelberg, Germany, 2008; chapter SQUASH—A New MAC with Provable Security Properties for Highly Constrained Devices Such as RFID Tags, pp. 144–157.
55. El Gamal, T. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In Proceedings of CRYPTO 84 on Advances in Cryptology, Santa Barbara, CA, USA, 19–22 August 1984; Springer-Verlag New York, Inc.: New York, NY, USA, 1985; pp. 10–18.
56. Paillier, P. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology-EUROCRYPT '99*; Stern, J., Ed.; Springer: Berlin/Heidelberg, Germany, 1999; Volume 1592, *Lecture Notes in Computer Science*, pp. 223–238.
57. Diffie, W.; Hellman, M.E. New Directions in Cryptography. Available online: http://www.cs.tau.ac.il/ bchor/diffie-hellman.pdf (accessed on 1 April 2013).
58. Harkins, D.; Carrel, D. RFC 2409—The Internet Key Exchange (IKE), 1998. Available online: http://tools.ietf.org/rfc/rfc2409.txt (accessed on 31 January 2013).
59. Blundo, C.; Santis, A.D.; Herzberg, A.; Kutten, S.; Vaccaro, U.; Yung, M. Perfectly-Secure Key Distribution for Dynamic Conferences. In Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara, California, USA, 16–20 August 1992; Springer-Verlag: London, UK, 1993; CRYPTO '92, pp. 471–486.
60. Liu, D.; Ning, P. Establishing Pairwise Keys in Distributed Sensor Networks. In Proceedings of the 10th ACM Conference on Computer and Communications Security, Washington, DC, USA, 27–30 October 2003; ACM: New York, NY, USA, 2003; CCS '03, pp. 52–61.
61. Perrig, A.; Szewczyk, R.; Tygar, J.D.; Wen, V.; Culler, D.E. SPINS: Security protocols for sensor networks. *Wirel. Netw.* **2002**, *8*, 521–534.

62. Chan, H.; Perrig, A.; Song, D. Random Key Predistribution Schemes for Sensor Networks. In Proceedings of the 2003 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 11–14 May 2003; IEEE Computer Society: Washington, DC, USA, 2003; SP '03, pp. 197–213.
63. Wong, C.K.; Gouda, M.; Lam, S. Secure group communications using key graphs. *IEEE/ACM Trans. Netw.* **2000**, *8*, 16–30.
64. Micciancio, D.; Panjwani, S. Optimal communication complexity of generic multicast key distribution. *IEEE/ACM Trans. Netw.* **2008**, *16*, 803–813.
65. Keoh, S.; Garcia-Morchon, O.; Kumar, S.; Dijk, S. *DTLS-based Multicast Security for Low-Power and Lossy Networks (LLNs)*; Technical report, IETF Internet Draft draft-keoh-tls-multicast-security-00; The Internet Engineering Task Force (IETF): Fremont, CA, USA, 2012.
66. Berkovits, S. How to Broadcast a Secret. In Proceedings of the International Conference on Theory and Application of Cryptographic Techniques (EUROCRYPT) Brighton, UK, 8–11 April 1991; Springer-Verlag: Brighton, UK, 1991; pp. 535–541.
67. Naor, D.; Naor, M.; Lotspiech, J. Revocation and Tracing Schemes for Stateless Receivers. In *Advances in Cryptology (*CRYPTO*)*; Kilian, J., Ed.; Springer: Berlin/Heidelberg, Germany, 2001; *Lecture Notes in Computer Science*, Volume 2139, pp. 41–62.
68. Ballardie, A. Scalable Multicast Key Distribution **1996**. The Internet Engineering Task Force (IETF): Fremont, CA, USA.
69. Lin, J.; Huang, K.; Lai, F.; Lee, H. Secure and efficient group key management with shared key derivation. *Comput. Stand. Interfaces* **2009**, *31*, 192–208.
70. Lee, P.; Lui, J.; Yau, D. Distributed collaborative key agreement and authentication protocols for dynamic peer groups. *IEEE/ACM Trans. Netw.* **2006**, *14*, 263–276.
71. Kim, Y.; Perrig, A.; Tsudik, G. Tree-based group key agreement. *ACM Trans. Inf. Syst. Secur.* **2004**, *7*, 60–96.
72. Briscoe, B. MARKS: Zero Side Effect Multicast Key Management Using Arbitrarily Revealed Key Sequences. In *Networked Group Communication*; Rizzo, L., Fdida, S., Eds.; Springer: Berlin/Heidelberg, Germany, 1999; *Lecture Notes in Computer Science*, Volume 1736, pp. 301–320.
73. Sarikaya, B.; Ohba, Y.; Moskowitz, R.; Cao, Z.; Cragie, R. *Security Bootstrapping Solution for Resource-Constrained Devices*; Technical report, IETF Internet Draft draft-sarikaya-core-sbootstrapping-05; The Internet Engineering Task Force (IETF): Fremont, CA, USA, 2012.
74. Jennings, C. *Transitive Trust Enrollment for Constrained Devices*; Technical report, IETF Internet Draft draft-jennings-core-transitive-trust-enrollment-01; The Internet Engineering Task Force (IETF): Fremont, CA, USA, 2012.
75. Garcia-Morchon, O.; Wehrle, K. Modular Context-Aware Access Control for Medical Sensor Networks. In Proceedings of the 15th ACM Symposium on Access Control Models and Technologies, Newark, NJ, USA, 20-22 June 2012; ACM: New York, NY, USA, 2010; SACMAT '10, pp. 129–138.
76. Hammer-Lahav, E. RFC 5849—The OAuth 1.0 Protocol, 2010. Available online: http://tools.ietf.org/rfc/rfc5849.txt (accessed on 31 January 2013).

77. Hardt, D. RFC 6749—The OAuth 2.0 Authorization Framework, 2012. Available online: http://tools.ietf.org/rfc/rfc6749.txt (accessed on 31 January 2013).