Universal Integration of the Internet of Things through an IPv6-based Service Oriented Architecture enabling heterogeneous components interoperability

Grant agreement for: Collaborative project

Grant agreement no.: 288445

Start date of project: January 1st, 2011 (36 months duration)

Deliverable D2.2

Distributed IPv6-based Security, Privacy, Authentication and QoS

Contract Due Date	30/09/2013
Submission Date	30/09/2013
Version	v.1.0
Responsible Partner	University of Luxembourg (UL)
Author List	F. Melakessou, T. Cholez, J. François, M.R. Palattella, A. Panchenko
Dissemination level	PU
Keywords	Internet of Things, IPv6, Routing, Security, Privacy

Project Coordinator: Mandat International (MI)

Sébastien Ziegler sziegler@mandint.org

Table of Contents

List of Tables

List of Figures

Executive summary

This deliverable reports the work done regarding security and privacy issues in the IoT6 architecture. QoS problematics have been addressed through the usage of a smart routing approach: this is the subject of an entire deliverable (D3.2). However, the title of deliverable D3.2 still mentions the QoS to be coherent with the Description of Work (DoW).

Therefore, D2.2 is focused only on security and privacy. While security has been a well-researched domain for many years ago, the current Internet is still facing many threats every day. Due to the more decentralized architecture of IoT, security is more challenging and so the goal of our work is to think proactively about security issues in IoT in order to already provide solutions or directions. Security both protects IoT service providers as well as their users and the deployment of such services is not only dependent on the service innovation but also on the confidence and the trust of the users, especially regarding their privacy and potential data leakage. The separation between security and privacy is actually very thin and not so well defined. In this deliverable, security is referred to techniques aiming at altering or protecting the network and services functions. Privacy will refer to more specific techniques aiming at identifying or protecting user specific data.

As envisioned in IoT6, the network structure can be divided into three parts: the Internet, the local networks and the sensor networks. Hence, we have developed solutions for the specific sensor network along with more traditional IP networks including both local networks and Internet even it is not our primarily objective.

For sensor networks, there are several solutions which have already been proposed and standardized in particular for enabling authentication, integrity and confidentiality. That is why with such solutions an analysis and testing of the latter have been achieved. Since encryption based techniques cannot avoid everything, i.e. a flooding attack, monitoring is a vital task of the network administrator. Nowadays, they are faced with new challenges. First, attacks are complex and may involve many steps with different protocols delayed in time. Second, alerts to verify, including false ones and raw data continues to increase drastically. Hence, a new scalable approach to rapidly highlight important changes in network related data has been designed.

As mentioned previously, QoS is now fully addressed in D3.2. However, D3.2 proposes the use of Content Centric Networking (CCN) for solving such issues. This is rather a new networking approach and therefore we decided to assess the latter in terms of security, since this is essential before envisioning deploying it. In fact, both works (CCN smart routing and security) have been done in parallel.

1 Overview

1.1 Purpose and scope of the document

The IoT6 research project aims at researching and exploiting the potential of IPv6 and IPv6-related technologies to develop an Open Service-oriented architecture overcoming the current Internet of Things fragmentation. This document is related to Task T2.2 on IPv6-based Security and QoS. This deliverable will consist of a report describing the appropriate security and privacy mechanisms for the IPv6 nodes of IoT6 architecture.

The IoT6 architecture implies a decentralized organization of the system. This Task is analysing the implication of the designed architecture in terms of security. It will research and address the implementation of features such as encryption, authentication, privacy, anonymity, and Quality of Service (QoS). This task will provide the appropriate security and privacy mechanisms for the IPv6 nodes of this architecture, including for the small and smart devices with constrained capacity such as 6LoWPAN devices. These mechanisms and protocols aim is to ensure the authenticity, integrity, and where needed confidentiality of the data passed between the IPv6 interconnected components on the Internet. It will take into account the decentralized nature of future IoT ecosystems, and the requirement to provide granularity in trust management and information sharing with other parties (inter-domain operability), including at the level of the resource repository. For that reason, optimized key exchange mechanisms, Public Key Infrastructure, optimized asymmetric key-based security (ECC), IPSec implementations for 6LoWPAN, the inclusion of an Identification Management (IdM) framework etc. are some of the specific subtasks to carry out in order to define the required functionalities and include the provision of the appropriate security infrastructure for the IoT6 connected components, including smart devices, web services, gateways, resource repository and multi-protocol boards.

Distributed Security, Privacy and Authentication with IPv6:

Traditional security mechanisms are based on a firewall approach, building a barrier between the inner network and the rest of the World. IoT6 will provide ubiquitous access through the Internet to smart things located in different regions, including mobile devices. It will take into account security, authentication and privacy aspects, by providing effective and adequate mechanisms. A key idea would be to distribute the firewalling functions at the level of the nodes. IoT6 will particularly explore the potential of IPSec, the Identification Management (IdM) Framework and similar technologies to provide global security, authenticity and anonymity. Those mechanisms will be applied to the nodes and to services such as resource repositories.

Adapting the Security Scheme to a Decentralized IPv6 Architecture:

Enabling the things to be connected directly on the Internet raises the issue of security and privacy. The “defragmentation” of the Internet of Things together with distributed intelligence will increase the risks of undesired connections, such as hacking. IoT6 will not be able to solve this problem in the frame of the STREP. However, it will identify the main risks associated to such a decentralized architecture and will explore different options to reduce those risks with adequate fire walling and authentication mechanisms. It will use existing tools, such as IPSec, tiny fire walling, and other available ones to distribute the security features and protect the strategic nodes. IoT6 will explore the integration of solutions such as low cost crypto primitives like those based on Elliptic Curve Cryptography (ECC) to support low cost/high performance secure identification/authentication devices, optimize HMAC authentication mechanisms for 6LoWPAN devices, and adapt key exchange protocols such as Diffie-Hellman.

1.2 Document outline

Based on its scope and purpose, the deliverable D3.2 has been organized in 4 main chapters.

The Introduction (Chapter 2) provides a general overview of the 6LoWPAN technology adopted by the IoT6 architecture.

Chapter 3 is focused on the security aspects. First of all, some of the protocols nowadays available for providing security at different layers of the protocol stack (i.e., IEEE802.15.4 MAC CCM*, IPSec, TLS, DTLS), are described, pointing out their weak aspects to be taken into account while building a secure IoT6 infrastructure. Some of the available solutions have been tested on real hardware. Then, a specific technique called Multi-dimensional aggregation Monitoring (MaM), has been suggested for performing large scale monitoring. The latter is a fundamental part of network management to enhance security. In fact, it helps to track abnormal behaviours (i.e., anomalies), coming from faults and/or attacks. Finally, a monitoring approach based on the new Content-Centric Networking (CCN) paradigm has been suggested in Chapter 3.

Chapter 4 illustrates privacy issues, mainly related to encryption techniques and to key management schemes. Highlights about traffic analysis are then illustrated: despite encryption, traffic analysis (patterns of communication, in detail: direction, frequency, size, visibility to an eavesdropper and intermediate nodes) can violate the privacy of the data flow in the network. Chapter 5, related to Quality of Service (QoS), simply points to deliverable D3.2 that deals with smart routing solutions able to provide QoS support in the IoT6 architecture. Finally Chapter 6 provides the conclusion to this document.

2 Introduction

2.1 6LoWPAN Technology

An adaptation layer for IPv6 has been designed for IEEE802.15.4 Low-power and Lossy Networks (LoWPANs or LLNs) on simple embedded devices. LLNs are by definition constrained environments with special characteristics such as small packet size, low bandwidth (IEEE802.15.4 PHY) and low cost devices deployed in large scales. These embedded devices are low-power, small and cheap. They also use short-range, low-power wireless radios which have limited data rates, frame sizes and radio duty cycles.

6LoWPAN architectures are often based on host and router nodes connected to one or several edge routers which share a common IPv6 address prefix, distributed by the edge router and thereafter routers throughout the 6LoWPAN network. Three modes are available:

- Simple 6LoWPAN: nodes are connected through a single 6LoWPAN edge router to another IP network (as shown in Figure 1). Each node is identified by a unique IPv6 address, and is capable of sending and receiving IPv6 packets. Typically LoWPAN nodes support ICMPv6 traffic such as “ping”, and use the User Datagram Protocol (UDP) [78] as a transport protocol. The edge router (6LoWPAN Gateway) is in general directly connected to the Internet. It is responsible of 6LoWPAN compression and neighbour discovery. It also handles IPv6 forwarding on behalf of the nodes inside and outside the 6LoWPAN network. Reduced Function devices (RFDs) are simple nodes that are only capable to sense their environment and communicate data to Full-Function Devices (FFDs) [19].
- Extended 6LoWPAN: in this mode several edge routers are used and shared a common backbone link.
- Ad-hoc 6LoWPAN: this mode requires no infrastructure because the 6LowPAN is not connected to the Internet. For that purpose, one router must be configured as a simplified edge router, implementing Unique Local Unicast Address (ULA) generation and handling 6LoWPAN Neighbour Discovery registration. An IPv6 local prefix is advertised. As a consequence, there are no routes outside the 6LoWPAN network.

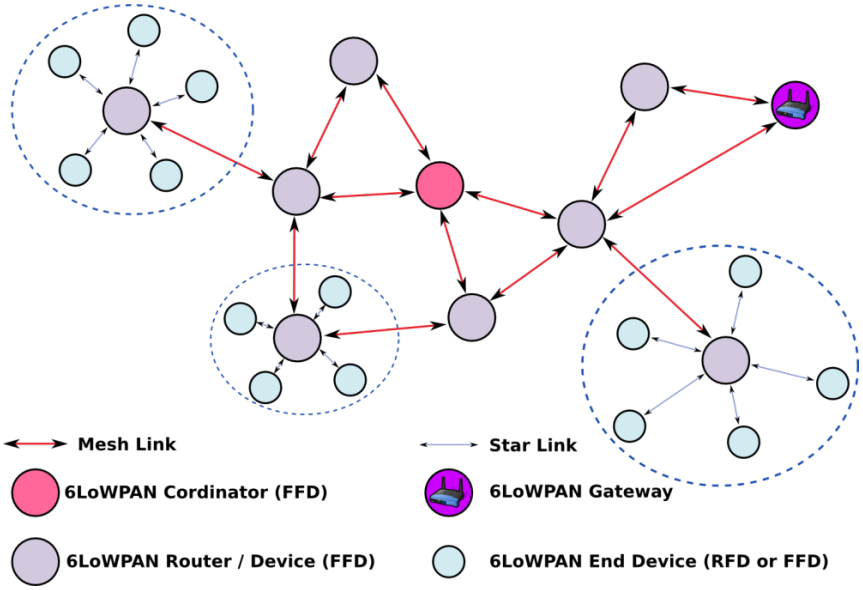

Figure 1: Simple 6LoWPAN network topology

In fact nodes register with an edge router through a Neighbour Discovery process that defines available interactions between hosts and routers on the same link. Nodes can be attached to more than one 6LoWPAN at the same time (multi-homing). They can be mobile. This will have a direct impact on the network topology that can also change due to the evolution of wireless channel conditions. Nodes can communicate with other IP nodes in an end-to-end manner. In fact each 6LoWPAN node is identified by a unique IPv6 address, and is capable of sending and receiving IPv6 packets. In general, 6LoWPAN nodes support ICMPv6 traffic such as ping, and use UDP as transport, due to their limitation in payload and processing capabilities.

IP addressing with 6LoWPAN works just like in any IPv6 network. IPv6 addresses are typically formed automatically from the prefix of the 6LoWPAN and the link-layer address of the wireless interfaces. For 6LoWPAN technologies, a direct mapping between the link layer address and the IPv6 address is used for achieving compression. Adaptation between full IPv6 and the 6LoWPAN format is performed by edge routers.

The 6LoWPAN gateway supports by definition two protocol stacks, i.e., IP and 6LoWPAN stacks, as shown in Figure 2. The adaptation layer is located between the IEEE802.15.4 link layer and the IP layer. It is the main component of 6LoWPAN. It enables TCP/IP communications above it. In fact the adaptation layer is mandatory as the size of IEEE802.15.4 frames do not permit to use conventional IPv6 packets. Within a 6LoWPAN network, IEEE802.15.4 frame size is limited to 127 bytes. The Maximum Transfer Unit (MTU) within IPv6 networks rates 1280 bytes. So header compression, fragmentation and re-assembling are handled by the adaptation layer. The adaptation layer also supports routing between the 6LowPAN network and the remaining Internet.

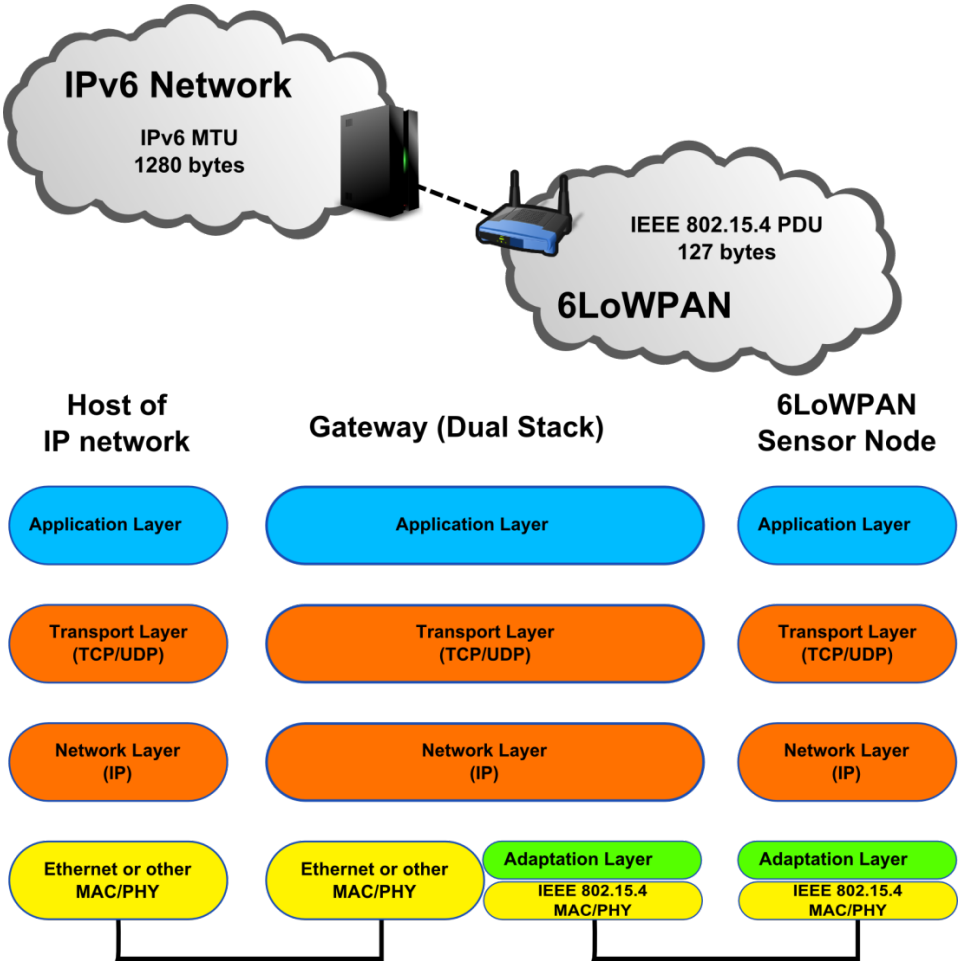

Figure 2: Dual Stack Gateway in a 6LoWPAN network

6LoWPAN applications most often involve completely autonomous devices and networks which must auto-configure themselves. Bootstrapping (channel setting, default security key and address settings), performed by the link layer, enables basic communication between nodes within a defined radio range. After this initialization, devices can operate single-hop communications. Afterwards 6LoWPAN Neighbour Discovery (ND) [79] is used to bootstrap the whole network. In fact the original IPv6 ND supports most basic bootstrapping and maintenance issues between nodes on IPv6 links. But without any modification, IPv6 ND remains not suitable for 6LoWPAN networks. Thus 6lowpan-nd has been designed to describe auto-configuration and the operation of hosts, routers and edge routers. It is an important part of the bootstrapping process. A node uses ND to discover other nodes on the same links, to determine their link addresses, to find routers and to maintain reachability information about the paths to neighbours that the node is actively communicating with. The edge router updates a registry of the 6LoWPAN nodes in order to simplify IPv6 operations across the network and reduce the amount of flooding.

Contrary to WLAN commissioning where SSID (Service Set Identifier) and security information (key material) are used, a 6LoWPAN device can assume that it is connected to its network when the security parameters and keying material match. In that case Integrity check of incoming packets does not fail. As a matter of course, providing the network prefix, at the commissioning time would reduce the need for neighbour discovery router advertisements as nodes will automatically be aware of the 6LoWPAN where they are attached. The edge routers include the context information in their Route Advertisement (RA), making it available to all first hop routers, which disseminate it further down on the topology and so on.

Addressing is also an important process that aims at assigning the IP address of 6LoWPAN nodes. During the design of IEEE802.15.4, a 64-bit MAC address is assigned to each device, and afterwards it can be used as EUI-64. This identifier has to be unique and can also be used as device ID by applications. A simple mechanism can be used for address configuration: Stateless Address Auto-configuration. In fact it is based on Interface Identifier IID that is performed with each interface on the link, ensuring uniqueness at least per link. This IID is combined with a prefix to form an address.

3 Security

One important aspect of setting up a 6LoWPAN is establishing and maintaining its security. Wireless networks by their nature are wide open to attackers, who may overhear and inject packets being exchanged, possibly using advanced antenna technology to mount their attacks from well outside the normal range of IEEE802.15.4 devices [19]. In order to allow a real take off of the Wireless Embedded Internet, security and privacy aspects needs to be properly taken into account. In details, the following security objectives should be fulfilled:

- **Confidentiality**. Data cannot be overheard by unintended listeners, i.e., remains secret except for the authorized participants of the conversation. This is actually not possible in a literal sense, but can be achieved in effect by making the information unintelligible by cryptographic **encryption**.
- **Integrity**. Data cannot be altered by unauthorized parties, i.e., the data that is being received by an authorized party is identical to the data that was sent by another party. In a digital world, tampering with a message may not leave any detectable traces, so integrity is often achieved by adding cryptographic **integrity checks** to messages. A message integrity check often consists in a message **authentication check**, i.e., in checking if the message is actually originating from the source that it claims to be.
- **Availability.** Data should be available to the right party at the time they are needed. In other words, the system is not subject to *denial of service attacks*.

Once the security objective have been defined based on the specific application requirements, it is necessary to understand the *threat model*, i.e., (1) what the attacker is going to be able to do that might work against the security objectives; and (2) the level of benefit that the attacker may derive from subverting a security objective.

For wireless systems such as 6LoWPAN, the threat model assumed for Internet security protocols, the Dolev-Yao model [Dolev81-RFC3552] can be adopted: the attacker is assumed to have practically complete control over the communication channel. The attacker can read any message, remove and change existing message, as well as inject new messages. Without *cryptographic support*, there is no way to protect messages from reading or to detect a message that has been tempered with.

The Internet threat model assumes that the end systems have not been compromised, mainly because it is very hard to maintain full security if that assumption cannot be made. However, the small size and the distributed nature of the LoWPAN nodes create a rather significant threat here: in many LoWPAN deployments, it will be relatively easy to physically obtain and control at least one node in the network: the low-cost requirement will limit the level to which the node can be made tamper-proof. Measures need to be taken to control the damage. In particular, it is fundamental that the protection of the entire network does not depend on the integrity (and confidentiality of the memory) of each and every single node. This can create hard problems, and it is therefore important to find the right trade-off between potential damage and the cost of providing and maintaining the security.

3.1 Security Highlights in IPv6 networks

Security can be handled by TLS (Transport Layer Security) or DTLS (Datagram TLS). IPSec (IPv6) is possible at the network layer but it consumes a lot of resources. Limitations in 6LoWPANs prevent the use of the full IPSec suite. The management of cryptographic keys using the minimum payload, and the limitation of exchanged messages between nodes are required. LSEND (Lightweight Secure Neighbour Discovery) is an extension of the protocol SEND that permits to secure the Neighbour Discovery mechanisms in 6LoWPAN networks (6LoWPAN layer). AES enables to secure the link layer. Link-layer security inside 6LoWPANs, based on the IEEE 802.15.4 128-bit AES encryption, provides some protection. Authentication can be done in respect with the resurrecting duckling model.

Heavy Security protocols needs to be avoided as much as possible due to the intrinsic low power and computational resources of 6LoWPAN smart objects. Symmetric encryption is privileged. Node and network limitations in 6LowPAN prevent the use of the full IPSec suite, transport layer ("socket") security or the use of sophisticated firewalls on each node. However DTLS can be used on low power nodes, and it provides a suitable level of end to end security between web clients and embedded web servers.

If 6LoWPAN is selected as the communication protocol for the wireless sensor network, DTLS, LSEND and AES encryption are the related security components that the IoT6 architecture must use.

AES symmetric encryption is used for confidentiality and integrity. AES/CCM is a quite efficient and very secure algorithm as long as the same nonce never occurs twice with the same key. So, many applications will require rekeying within the lifetime of the 6LoWPAN network.

3.1.1 IEEE802.15.4 Data link security

The IEEE802.15.4 standard provides link layer security services. It has three operational modes: (i) unsecured; (ii) an Access Control List (ACL) mode; and (iii) secured mode.

In the unsecured mode, as the name implies, no security services are provided. In the ACL mode the device maintains a list of devices with which it can communicate. Any communication from devices not on the list is ignored. However, it has to be noticed that this mode offers no cryptographic security so it is trivial for the message source address to be spoofed.

Finally, the IEEE802.15.4 secured mode is designed to facilitate the use of *symmetric key cryptography* in order to provide data confidentiality, data authenticity and replay protection. It is possible to use a specific key for each pair of devices (link key), or a common key for a group of devices. However, the mechanism used to synchronize and exchange keys are not defined in the standard, and left to the applications.

The degree of frame protection can be adjusted on a frame per frame basis. In addition, secure frames can be routed by devices that do not support security.

In order to protect the privacy and/or authenticity of messages, IEEE802.15.4-2011[19] uses a counter with **cipher block chaining mode*** (CCM*) cipher suite mode, an extension of CCM mode defined in ANSI X9.63.2001. The CCM* mode is interoperable with the CCM mode, while at the same time it goes beyond some of its limitations.

The CCM mode is a mode of operation that operates on block-ciphers with a 128-bit block size and involves a particular combination of the Counter (CTR) mode of operation and the Cipher-Block Chaining (CBC) mode of operation, using a single key. The CCM mode allows for variable-length authentication tags (from 32-bits to 128-bits), thus allowing varying degrees of protection against unauthorized modifications. The CCM mode allows quite

efficient implementations, due to the fact that one only needs to implement the encryption transformation of the underlying block-cipher (and not the decryption transformation) and due to its reliance on a single key, rather than multiple keys, to provide confidentiality and authenticity services. This being said, the CCM mode has also some disadvantages, including the following:

1) While the original CCM mode provides for data authentication and, possibly, confidentiality, it does not provide for confidentiality only. This is unfortunate, since not all implementation environments call for data authenticity (e.g., when data authenticity is provided by an external mechanism). It would be advantageous to have a single mechanism that could provide for all suitable combinations of confidentiality and authenticity, rather than a strict subset hereof.

2) The original CCM mode is known to be vulnerable to specific attacks, if used with variable-length authentication tags rather than with fixed-length authentication tags only. Thus, the original CCM mode can only be securely used with the same key in settings with fixed-length authentication tags. This is unfortunate, since support for variable-length authentication tags is useful in constrained implementation environments, such as secured wireless sensor networks, where applications on a device might have different protection requirements, but would have to share the same key, due to resource constraints.

The CCM* mode extends the definition of the original CCM mode, such as to provide for confidentiality-only services, in addition to the other security service options already offered. Moreover, the CCM* mode adapts the original CCM mode so that the resulting mode can be used securely with variable-length authentication tags, rather than fixed-length authentication tags only. Thus, the CCM* mode takes away the disadvantages of the CCM mode pointed out above.

At the same time, the specification of the CCM* mode is such that it is compatible with the CCM mode.

3.1.1.1 CCM* Security Transformations

CCM* takes as input a string “a” to be authenticated using a hash code and a string “m” to be encrypted, and delivers an output cipher text comprising both the encrypted version of “m” and the CBC message authentication code (CBC MAC) of “a”. Figure 3: Overview of CCM* security transformations. Figure 3 shows the transformations employed by CCM*, which uses the AES block cipher algorithm E.

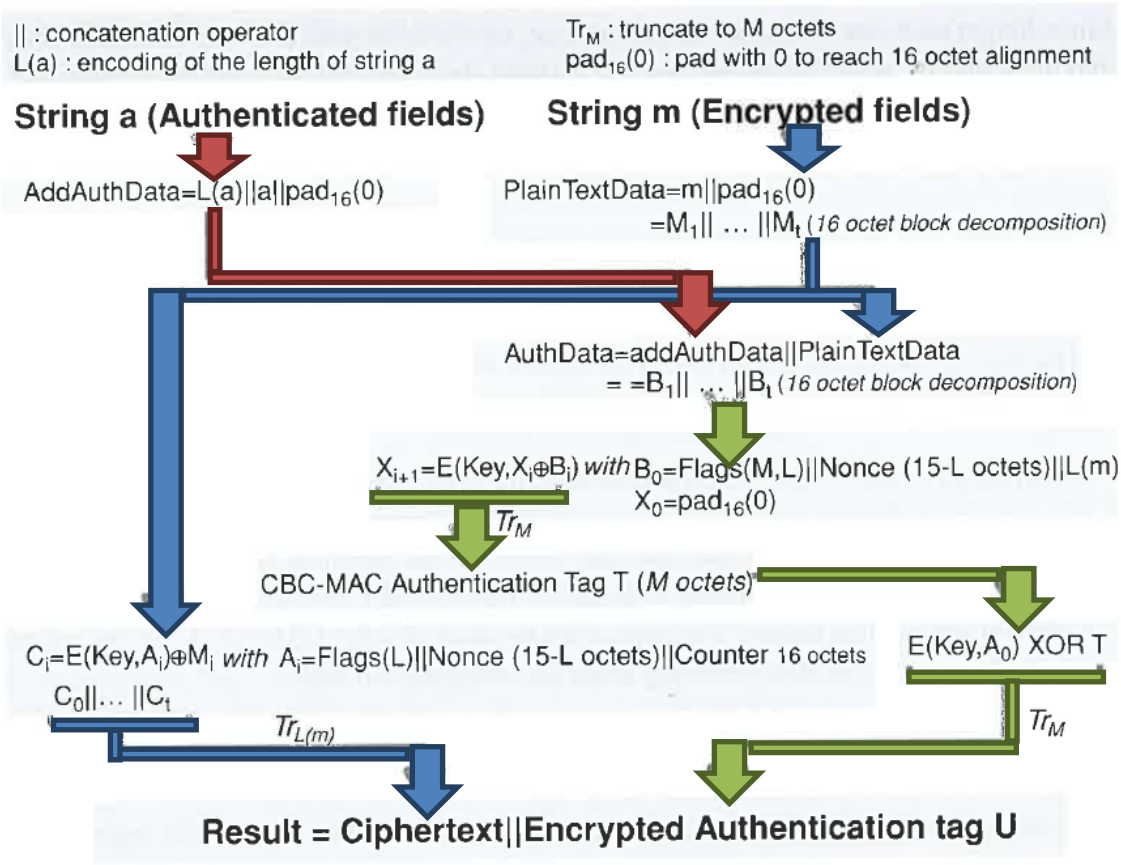

Figure 3: Overview of CCM security transformations.*

In the case of IEEE802.15.4, L = 2, and the nonce is a 13-bytes field composed of the 8-bytes address of the device originating the frame, the 4-bytes frame counter, and the 1-byte security-level code.

3.1.1.2 The Auxiliary Control Header

The required security parameters are contained in the auxiliary control header, which is composed of a security control field (1 byte), a frame counter (4 bytes) ensuring protection against replay attacks, and a key identifier field (0/1/5 or 9 bytes).

The first 3 bits of the security control field indicate the security mode for this data frame (Table 1), the security mode determines the size of M in the CCM* algorithm (0,4,8 or 16 bytes), and the data fields included in the “a” and “m” strings used for the computation of the final cipher text (security attributes). The next 2 bits indicate the key identifier mode (listed in Table 2), and the remaining bits are reserved.

Security control field	Security attributes	Data confidentiality (data in "m" string)	Data authenticity (data in "a" string)
000	None	OFF	No
001	MIC-32	OFF	MHR, Auxiliary security header, Non-payload fields, Unsecured payload fields
010	MIC-64	OFF	
011	MIC-128	OFF	
100	Encrypted fields	Unsecured payload fields	No
101	Encr. Fields + MIC-32		MHR, Auxiliary security header, Non-payload fields
110	Encr. Fields + MIC-64		
111	Encr. Fields + MIC-128		

Table 1: Security Control field codes.

Key identifier mode	Description: Key determined …	Key Identifier field length
00	Implicitly from the originator and the recipient of the frame	0
01	from the 1-byte Key-index subfield of the Key identifier field, using the MAC layer default Key source	1
10	Explicitly from the 4-bytes Key source subfield, and the 1-byte Key index subfield of the Key identifier field (part of the auxiliary security header)	5
11	Explicitly from the 8-bytes Key source subfield, and the 1-byte Key index subfield of the Key identifier field (part of the auxiliary security header)	9

Table 2: Key identifier mode codes

It has to be noted that the IEEE802.15.4 is a very constrained protocol. In fact, it does not provide any fragmentation and reassembly mechanism. As the maximum packet size is 127 bytes, and the MAC headers and FCS take 6 and 19 bytes, respectively, applications need to be careful when sending unsecured packets larger than 108 bytes. Most applications require security: the security headers add between 7 and 15 bytes of overhead, and the message authentication code between 0 and 16 bytes. In the worst case, only 77 bytes are left to the application.

3.1.1.3 Key Selection

IEEE802.15.4 does not handle distribution of keys. The interface between the MAC layer and the key storage is a key lookup function, which provides a lookup string parameter that is used as an index to retrieve the appropriate key.

The look up material provided depends on the context, as shown in Table 3.

Addressing Mode	Sender lookup data (based on destination addressing mode)	Receiver lookup data (based on source addressing mode)
Implicit	Source PAN short or extended address	Destination PAN short or extended address
Short	Destination PAN and destination node address	Source PAN and destination node address
Long	Destination node 802.15.4 8 bytes extended address	Source node 802.15.4 8 bytes extended address

Table 3: Addressing Mode and related Look up material

With implicit key identification (KeyIdMode = “00”), the lookup data is based on the IEEE802.15.4 address. The design implies that, in general, the sender indexes its keys according to destinations, and the receiver indexes its keys according to the sources.

With explicit key identification, the lookup data is composed of a key source identifier, and a key index. The design implies that the key storage is organized in several groups called key sources (one of which is the macDefaultKeySource). Each key source comprises several key identified by an index.

The CCM standard specifies that a given key cannot be employed to encrypt more than 261 blocks; therefore the applications using IEEE802.15.4 should not only assign keys, but also change them periodically.

3.1.1.4 IEEE802.15.4 Security open issues

There have been some problems found with some of the IEEE802.15.4 optional security modes and with the feasibility of supporting different keying models but these limitations can be overcome by higher level protocols. The latter has to guarantee keying models without the use of much extra energy. Another issue rises up from the fact that no explicit key management scheme has been proposed in the standard.

Finally, as identified in [20], a major vulnerability is the unsecured ACK packet which renders most security measures at MAC layer useless. Notably, as the ACK frame integrity is not protected, it opens the door for a malicious node to prevent a legitimate device from receiving a particular frame, which is possible by forging an ACK using the unencrypted sequence number from the data frame and sending it to the source while creating enough interference, in order to prevent the legitimate receiver from receiving the frame. It hence allows for doS in form of ACK spoofing, which can be prevented by securing the ACK.

3.1.2 IPSec

Even though the best link layer (L2) mechanism is adopted in a LoWPAN, the data is no longer protected once it leaves the link. This makes the data vulnerable at any point that is responsible for forwarding it at the network layer (L3), or on any link that has lesser security. Moreover, an attack on the network layer might be able to divert data onto a path that contains additional forwarding nodes, controlled by the attacker.

Therefore, end-to-end security that protects the conversation along the entire path between the communicating nodes is an important element of any robust security system. This requirement is actually a banner feature in the development of IPv6.

IPSec is a network layer security suite that provides confidentiality and integrity at the IP layer [23]. IPSec was originally developed for IPv6, but it has been retrofitted to work for IPv4 as well.

IPSec has two main components: packet formats and related specifications that define the confidentiality and integrity mechanisms for the actual data, and a key management scheme called Internet Key Exchange (IKE), [RFC2409], updated by IKEv2 [25]. IPSec/IKE can be implemented in a host-to-host transport mode, or a network tunnel mode, and supports a variety of cryptographic algorithms (HMAC-SHA1, tripleDES-CBC, and AES-CBC).

IPv6 mandates the use of IPSec, but the feasibility of using IPSec on low-power smart object networks is still a matter of debate, mostly because of the increased power and RAM requirements imposed by cryptographic algorithms employed. Due to the relatively complex set of protocols that constitute IKE, IPSec is generally considered a poor fit for the requirements of LoWPAN. However, IPSec can be successfully implemented on smart objects provided that algorithms and key sizes are chosen with care [26].

IPSec defines two packet formats for cryptographically protected data: the IP Authentication Header (AH) [RFC4302] which provides integrity protection and authentication only, and the IP Encapsulating Security Payload (ESP) [27], which combines this function with confidentiality protection through encryption.

AH got a bad name with IPSec implementations for IPv4, as it protects not only its payload but also the address in the enclosing IP packet header. This detects and rejects the tampering with IP address performed by NATs, which are essential to the operation of IPv4 networks. Although AH is perfectly useful in the NAT-free IPv6 environments, this problem has led most IPSec implementers to focus on ESP.

3.1.2.1 IPSec Encapsulating Security Payload (ESP)

Figure 4 shows the ESP packet format. In contrast to other IPv6 extension headers, which simply precede their payload, ESP *encapsulates* it: this approach is somehow logical, given that the payload is encrypted. The part of the packet format marked "C" on the right is encrypted (confidentiality protected), while the larger part marked "I" is subject to integrity protection.

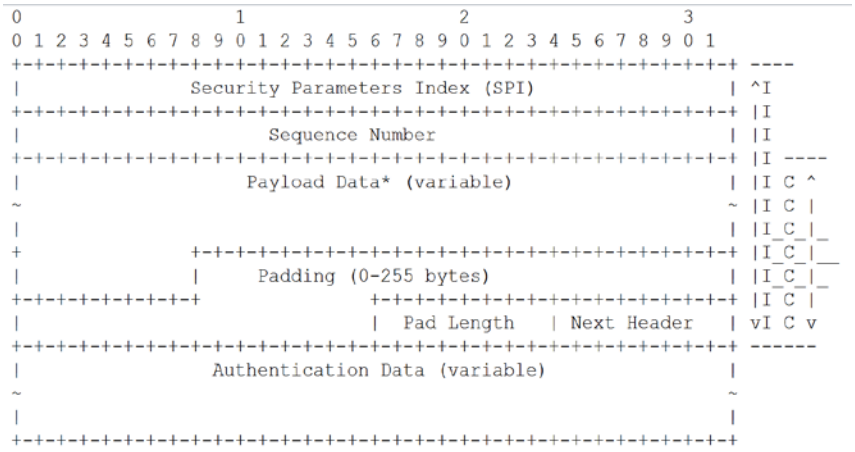

Figure 4: Encapsulating Security Payload (ESP) packet format

Keys and other information, such as which encryption algorithms are used, are stored at the end points. In details, the *security parameter index* (SPI) identifies the specific security parameters, including the keying material, used for this direction of the communication. A specific set of keys is called a *Security Association* (SA). Every packet belongs to a specific SA. Different SA may use different encryption algorithms. To set up an SA, IPSec uses the key management protocol IKE. For unicast packets, the SPI is an identifier of local significance to the receiver, i.e., it is assigned by the receiver in a way that facilitates its local processing of the incoming ESP packets.

IPSec makes use of both symmetric and asymmetric encryption mechanisms. For smart low-power devices, symmetric mechanisms are preferred because of their lower computation complexity. In detail, for smart objects equipped with hardware AES acceleration, which is common in many IEEE802.15.4 devices, it is possible to let IPSec take advantage of the AES acceleration. By using AES encryption as part of the IPSec SA, the device achieves a level of encryption performance that is impossible with software-only mechanisms.

The sequence number is an unsigned 32-bit number that increases by one for each packet sent on the security association; it can also be the lower 32 bits of a 64-bit sequence number kept in the security association. The payload data, together with the trailer consisting of the padding, the pad length and the next header field, are the part of the packet that is encrypted. Note that Figure 4 shows the unencrypted/decrypted version of the data.

The next header field specifies how the decrypted data is to be interpreted by the receiver. It can identify the payload as an IP packet (tunnel mode) or as some transport data such as an UDP header followed by a UDP payload (transport mode). Tunnel mode is most useful for security gateways (VPN gateways); transport mode is a more compact way to obtain end-to-end security as no second IP header needs to be sent.

The padding (the length of which is given by the eight-bit pad length field) can be added to round up the length of the payload and trailer (including pad length and next header) to a multiple of four bytes, which is the minimum required alignment of the beginning of the ICV field. Actually the encryption algorithm might pose more stringent alignment requirements,

such as a full AES block of 16 bytes for modes such as AES—CBC [28]; this is not an issue with streaming modes such as the AES-CCM most likely used in LoWPANs.

Finally, the integrity check value (ICV) is used with the other information in the ESP header, payload and trailer to check the integrity of the packet. The length of the ICV is defined in the security association.

3.1.2.1.1 ESP with AES/CCM

LoWPAN nodes are likely to have hardware for AES/CCM encryption, decryption and integrity check processing. This hardware is generally available in a way that allows software above the link layer to use it, making AES/CCM for ESP [29] the obvious candidate for the cryptographic suite used to achieve end-to-end confidentiality and integrity/authentication.

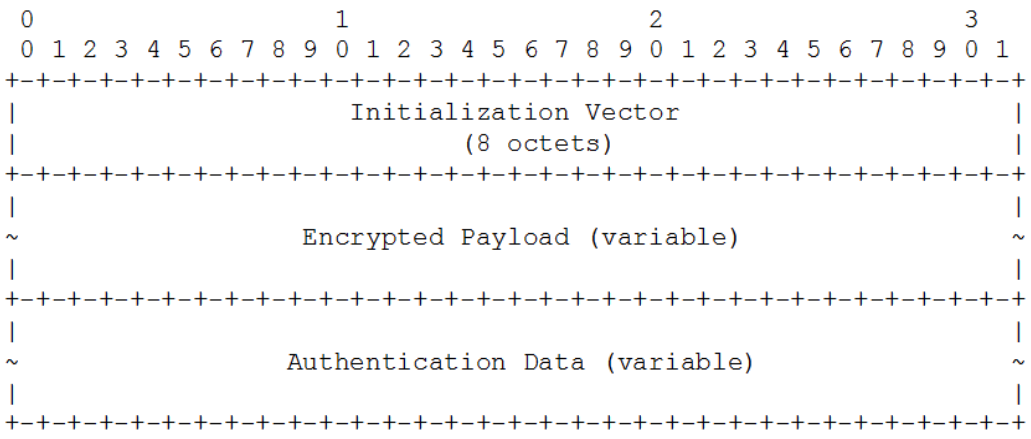

Figure 5: ESP payload encrypted with AES/CCM

Figure 5 shows how the encrypted payload and the ICV look with AES/CCM. The encrypted payload starts with an explicitly transmitted initialization vector (IV) of eight bytes. This is preceded by three bytes of salt in the security association to produce an 11-bytes nonce going into the CCM algorithm. The encrypted payload follows. Note that the payload has been fitted with a trailer before encryption, appropriately padded to a multiple of four bytes. Finally, the ICV is appended. [29] provides for ICV sizes of eight bytes as well as optionally 12 bytes; which of these values is used needs to be defined in the security association.

ESP with AES/CCM is not necessarily too heavy-weight for end-to-end encryption and integrity checking between a LoWPAN and its correspondent nodes. With most IEEE802.15.4 capable chips, the processing overhead is as small as it possibly could be. The per-packet overhead could be a bit less with 1-4 bytes of overhead for padding (including pad length) and eight bytes for explicitly transmitted initialization vector (IV), but it is not too bad, if solid end-to-end cryptographic protection is indeed needed. If more efficiency is really needed, a special version of AES/CCM transform could be defined that creates even less overhead, or preferably some special form of stateless header compression could be added to LOWPAN_NHC, e.g., by deriving the initialization vector from other information present in the MAC-layer packet.

3.1.3 TLS

Transport layer Security (TLS) provides an end-to-end secure channel between two network end points. It provides confidentiality and integrity as mechanism for authentication of the communication end points [30].

TLS was originally developed under the name Secure Sockets Layer (SSL) by Netscape Corporation for the Netscape web browser, but was later standardized by the IETF as TLS.

TLS consists of several different layers and protocols. At the lower layer, a symmetric encryption algorithm is used to provide confidentiality and integrity. To establish a key for use in the symmetric encryption algorithm, TLS first perform an authentication combined with a secure key exchange protocol. The authentication can be either unilateral, meaning that only one of the connection end points is authenticated, or bilateral, where both communication end points are authenticated.

Before initiating the authentication phase, the TLS end points engage in a protocol negotiation phase. The end points use this phase to decide what encryption protocol to use for the reminder of the connection. TLS supports several encryption protocols.

In the authentication phase TLS makes extensive use of asymmetric cryptography. For low-power devices this security mechanism is inappropriate. Therefore, some effort has been put to provide more lightweight cryptographic algorithm to achieve end-to-end security for computationally constrained microprocessors [31][32].

TLS is most often applied between the sink and the rest of the network, but not within the network of smart objects themselves. That is because TLS require a reliable transport mechanism (i.e., TCP), and low-poor networks, and 6LoWPANs sometimes do not support TCP.

3.1.4 DTLS

Datagram Transport Layer Security (DTLS)[RFC6347] is a protocol used to secure datagram traffic for client/server applications. DTLS is composed of a Record Protocol that carries other protocols such as Handshake, Alert, and application data. The initial Handshake authenticates the server and optionally the client using a Public Key Infrastructure (PKI).

Recently, the new connection-less lightweight Constrained Application protocol (CoAP), specifically designed for LLNs, and which easily translates to HTTP for integration with the web, has proposed to use DTLS as a security protocol for automatic key management and data encryption and authentication. The CoAP protocol with DTLS support is known as secure-CoAP (CoAPs).

Figure 6 shows an IoT network scenario, including a 6LoWPAN that consists of CoAPs enabled nodes and a 6LoWPAN Border Router (6LBR). Messages within 6LoWPAN travel in compressed form. The 6LBR acts as a bridge between 6LoWPAN and the conventional Internet, thus allowing for the access to the CoAP/6LoWPAN devices from anywhere in the Internet. This implies that the 6LBR is an authenticated part of the constrained network, achieved by prior bootstrapping mechanism in the LLN. As shown in Figure 6, CoAPs-enabled devices can securely communicate with Internet hosts such as laptops, smartphones that support DTLS.

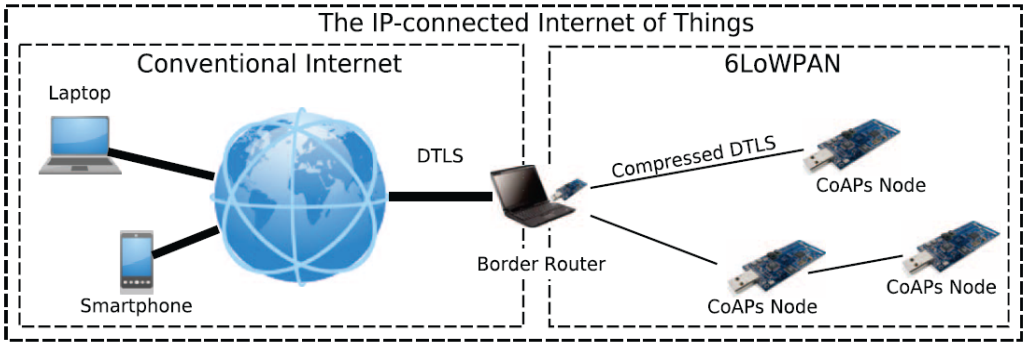

Figure 6: An IoT setup that uses CoAP to secure communication between sensor nodes in 6LoWPANs and hosts in the Internet

3.1.4.1 Security consideration

LLN devices, being resource-constrained devices, are prone to flooding and resource exhaustion attacks. An attacker is luckier to be an Internet host which possesses the addresses of the target devices, and attempts to start multiple end-to-end handshakes with the LLN devices in order to disrupt the operation of the LLN.

The DTLS handshake as it is, would allow anyone to initiate a handshake, thus causing the victim device to repeatedly create a new DTLS context when a handshake request is received. Consequently, the attacker can exhaust the resources of battery-powered devices in the LLN. For that reason DTLS introduced a stateless cookie mechanism in order to prevent this attack. However, it is recognized that whenever the LLN device has to send a message, energy is consumed. Therefore, any undesirable and malicious network traffic could flood the network, utilizing not only the resources of the destination node, but also routing nodes and other nodes in the vicinity that receive the message on the physical layer.

CoAP messages usually contain commands to the nodes to do a specific task, e.g., 'Switch on the lights' or 'Send update messages'. Replaying these messages can disrupt the behaviour of the nodes, the 6LoWPAN and might lead to a complete system failure. Without filtering on the 6LBR, the replayed packet can greatly affect the performance of the network similar to the consequences of flooding attacks.

DTLS has already defined a mechanism to detect message replay. However, before the node detects that a message is replayed, it has to receive, process and possibly forward the message between nodes. All these operations are energy consuming, and it has the same effect caused by a flooding attack. Given that the bandwidth in the LLN is limited, the fewer (unnecessary) packets are sent the better it is.

Lastly, a mechanism to identify amplification attacks is needed. Amplified packets are quite huge, resulting in fragmentation on IP layer in the 6LBR, consuming energy in the receiving nodes on the path and the node itself by de-fragmenting the packet [37].

3.1.4.2 Integrating Compressed DTLS into 6LoWPAN

DTLS was designed for the Internet and not for resource constrained Internet of Things (IoT) devices; therefore, it is a heavyweight security protocol. IoT devices can use the 6LoWPAN protocol to compress the long IP layer headers. By exploiting 6LoWPAN compression capabilities, it is possible to compress the DTLS headers and messages.

The 6LoWPAN standard defines IP Header Compression (IPHC) for the IP header and the Next Header Compression (NHC) for the IP extension headers and the UDP header.
The 6LoWPAN standard does not provide ways to compress the UDP payload and the layers above. However, recently [34] has been proposed as a plug-in for 6LoWPAN which can be used to compress the UDP payload.

DTLS as being part of the UDP payload can be compressed using 6LoWPAN-GHC. S. Raza et al. have exploited this feature in [36].The proposed ID bits in the NHC for UDP-GHC are used to differentiate NHC for UDP from NHC for UDP-GHC. The ID bit 11010 in the NHC for UDPGHC, as defined in the [34], indicates that the UDP payload is compressed with 6LoWPAN-GHC. Once the UDP-GHC is defined it is possible to compress DTLS by providing GHC for the DTLS messages.

When the 6LoWPAN-GHC for DTLS is completed they use the STOP code, set to 10010000 in [34], to indicate that the DTLS header compression is completed. They have defined 6LoWPAN-GHC for the Record header, Handshake header, Client Hello message, and Server Hello message. The same compression techniques can be used to compress the other Handshake messages. In detail, they have shown that using 6LoWPAN-GHC compression mechanisms it is possible to significantly reduce DTLS headers sizes, and save a significant number of bits. For instance, the Record header that is included in all messages when the DTLS is enabled can be compressed by 64 bits (62% of the Record header) for each message [36].

3.1.5 Analysis of Potential Candidates

3.1.5.1 Operating Systems

Specific Operating Systems (OSs) such as Contiki (www.contiki-os.org) and TinyOS (docs.tinyos.net) have been designed in order to efficiently work on embedded devices [5]. They intrinsically support the 6LoWPAN standard. The main goal of such operating systems is to optimally use limited resources of constrained sensors. These OSs are very simple in comparison to the ones found in normal computers. In fact, micro-controllers are only able to process one task at a time. As a consequence, OSs is based on a scheduling system. Thus the CPU resources can be shared between the different programs. In general, the application and the OS are compiled and deployed into a single binary. We present in the following sections the two main OSs used for the Internet-of-Things.

3.1.5.1.1 Contiki

Contiki is an open source operating systems created and developed at the Swedish Institute of Computer Science (SICS) by A. Dunkel et al [7]. Development is done by a team of experts from companies such as Atmel, Cisco, ETH, Redwire LLC, SAP, Thingsquare, and many others. Source code and information can be obtained within the Internet at the following web page: http://www.contiki-os.org.

Contiki permits connection to the Internet by small, low-cost, low-processing (mHz), low-power (mW) and battery-operated embedded devices with low communication bandwidth (kbits/s). It only needs a few kilobytes of code and a few hundred bytes of RAM. A typical system with full IPv6 networking with sleepy routers and RPL routing needs less than 10k RAM and 30k ROM. The core of Contiki is based on the kernel, drivers, different libraries, and the program loader. It is usually deployed into a single binary. All components of an application have a direct access to the underlying hardware. Contiki presents an event-driven kernel.

Processes can only be executed by the scheduler. Pre-emption is not handled by the scheduler, thus a process always run to completion in order to enable the consideration of new ones. Events can be asynchronous (in that case, an event is often served in a First In First Out manner, in respect with its chronological order) or synchronous (in that case, it is immediately processed by the event handler). Multitasking is provided.

Contiki supports many platforms equipped with different micro-controllers such as Atmel ARM, Atmel AVR, LPC2103, Microchip dsPIC, Microchip PIC32, TI MSP430, TI CC2430 and STM32w, etc. It permits to easily compile an application for any of the supported platforms. In fact, Instant Contiki contains complete and complex compiler tool chains. A simulator named Cooja permits the emulation of networks. Instant Contiki provides an entire development environment in a single download. Cooja permits the simulation of applications in large-scale networks. It also enables to run applications on fully emulated hardware devices.

Networking is ensured according to three network mechanisms, i.e. uIP TCP/IP (IPv4), uIPv6 (IPv6), and the Rime stack designed for low-power wireless networks. It provides a full IP network stack, with standard IP protocols such as UDP, TCP, and HTTP. It fully supports IPv6 and IPv4 standards, along with the recent low-power wireless standards, i.e. 6lowpan, RPL IPv6 multi-hop protocol and CoAP RESTful application-layer protocol. uIPv6 considers the RPL routing protocol for Low-power and Lossy IPv6 Networks (LLN) and the 6LoWPAN header compression and adaptation layer for IEEE 802.15.4 links. The Rime stack provides a set of primitives such as single-hop broadcast (sending a message to all neighbours), single-hop/multi-hop unicast (sending a message to a specified neighbour), network flooding, and address-free data collection, for low-power wireless systems. ContikiMAC enables nodes to reduce their power consumption. In fact nodes can run in low-power mode and still be able to receive and relay radio messages. The ContikiMAC radio duty cycling mechanism allows them to sleep between each relayed message to save power. In that case routers are named sleepy routers.

Contiki programs are based on protothreads, using features from multi-threading and event-driven models. In fact, event-handlers can be stopped, waiting for events to occur. Thus the kernel invokes the protothread of a process in response to an internal/external event. A Contiki process must always explicitly interact with the kernel at regular intervals. Contiki supports dynamic loading and linking of modules at run-time. Contiki applications are written in the standard C programming language. Examples are available inside the source code repository. A lightweight flash file system, named Coffee is included in Contiki. Then, programs can open, close, read from, write to, and append to files on an external flash equipped on the embedded device.

3.1.5.1.2 TinyOS

TinyOS is an open source BSD-licensed operating system project, created and developed at the University of California in Berkeley. It has been designed for low-power devices, such as those used in wireless sensor networks, ubiquitous computing, personal area networks, smart buildings, and smart meters. The TinyOS Alliance is maintaining this open source project. A worldwide community uses, develops, and supports this operating system. The last stable version released in August 2012 is TinyOS 2.1.2. TinyOS is now hosted at GitHub where the main development repository can be retrieved.

The operating system and applications are composed by components, i.e. commands, events and tasks. Commands ask a component to realize something. Events permit to highlight the completion of a command. When a task is posted, the currently running program will continue its performance. In fact a scheduler is responsible of the later execution of the posted Task. Components are connected to each other using interfaces. Interfaces allow components

exposing which commands they can send and which events they can support. An interface is composed by a number of commands and events. A configuration defines for each component interface which other component uses the provided interfaces and which component provides the interfaces through connections named wirings. TinyOS is non-blocking as it has one stack. Therefore, all I/O operations that last longer than a few hundred microseconds are asynchronous and have a callback. A TinyOS component can post a task. Tasks are non-pre-emptive and run in the First-In First-Out order. It is also possible to use a specific thread library, named TOSThreads.

TinyOS is based on a split-phase execution model. An operation request and the response after its completion are decoupled. Hardware and software modules are components that can run commands and later create events when the operations are completed. Tasks are executed by the scheduler. Each task is considered separately. Thus each task is performed until its completion that enables the service of the newt task and so on. In this case, memory is saved as the same stack is used for all tasks. If there are no more tasks to complete, the mote is set into a low-power sleep state until an interrupt wakes up the microcontroller. It is important to avoid long running tasks, as pre-empting is not supported by TinyOS. The hardware abstraction architecture (HAA) permits the creation of software independently from the platform if it is possible. The hardware interface layer (HIL) is the most platform independent level of device drivers, offering the common functionality to all devices using the common interfaces. The hardware adaptation layer (HAL) offers platform independent interfaces when possible and platform specific interfaces otherwise. The hardware presentation layer (HPL) provides the hardware functionality.

TinyOS provides interfaces and components for packet communication, routing, sensing, actuation and storage. TinyOS applications and OS are written in NesC [6]. The core of the TinyOS toolchain is the NesC compiler. Current implementations of the NesC compiler take all NesC files, including the TinyOS operating system, that belong to a program and generate a single C file. This C file can then be compiled by the native C compiler of choice for the target platform. The resulting small binary can then be deployed on the motes. TinyOS supports different hardware platforms, using ATMEL AVR family of 8-bit micro-controllers, Texas Instruments MSP430 family of 16-bit micro-controllers, etc. The compiled image of the application includes only the needed functions. The NesC language is optimized for constrained environments such as wireless sensor networks. When a command/event should be performed, the component's implementation of the command/event is selected with the wiring in configurations.

3.1.5.1.3 Hardware Platform and Operating Systems used for Tests

We have used 802.15.4 TelosB mote modules manufactured by the Spanish manufacturer Advanticsys (www.advanticsys.com). The CM5000 TelosB sensors is an IEEE 802.15.4 compliant wireless sensor node based on the original open-source TelosB/Tmote Sky platform design developed and published by the University of California, Berkeley. The included sensors measure temperature, relative humidity and light. Each node is equipped with a TI MSP430F1611 Microcontroller, a CC2420 RF Chip, User & Reset Buttons, 3 Leds, an USB Interface and 2xAA Batteries.

Figure 7: CM5000 mote

CM5000 motes support TinyOS and Contiki. Advanticsys provides a Vmware Image in order to start working with TinyOS. Another TinyOS Virtual Machine is also available and was used during the 7th SenZations Summer School on IoT and Applications, sponsored by the IoT6 project (3-7 September 2012, Drvengrad, Mecavnik, Serbia). Instant Contiki is an entire Contiki development environment, based on an Ubuntu Linux virtual machine that runs in VMWare player or VirtualBox, and has Contiki and all the development tools, compilers, and simulators used in Contiki development already installed.

Inside the TinyOS virtual machine, the path $TOSROOT links to the TinyOS root directory:

- **apps**: Many sample applications are available which test a part of the system (Blink, CoapBlip, Oscilloscope, PpppRouter, Sense, tosthreads, etc.)
- **doc**: Documentation and TinyOS Programming Manual
- **support**: Non-nesC Code for using TinyOS nodes
- **tools**: TinyOS specific tools and scripts
- **tos**: TinyOS source code (nesC, C)

Inside the Instant Contiki virtual machine, the root directory is ~/contiki-2.6:

- **apps**: Many sample applications (erbium, er-coap-03, er-coap-07, er-coap-12, er-coap-13, json, rest-coap, etc.)
- **core**: Contiki source code (C)
- **cpu**: Set of supported micro-controllers (arm, avr, msp430, x86, etc.)
- **doc**: Documentation and Contiki Programming Manual
- **examples**: Many sample applications are available which test a part of the system (collect, er-rest-example, ipv6, multi-threading, rest-example, rime, sky, etc.)
- **platform**: Set of supported platforms (avr-raven, econotag, iris, micaz, sky, wismote, z1, etc.)
- **tools**: Contiki specific tools and scripts

Three CM5000 motes are available (motelist):

- Mote 1: Reference MFV69HQG Description FTDI MTM-CM5000MSP
- Mote 2: Reference MFV6KBXA Description FTDI MTM-CM5000MSP
- Mote 3: Reference MFV69BLV Description FTDI MTM-CM5000MSP

3.1.6 Test of Security Solutions for IoT

There exist many alternative solutions that provide security during the deployment and operation of embedded devices within a wireless sensor network. We describe in the next section some of them, based on TinyOS and Contiki.

3.1.6.1 AES in TinyOS

Security can be ensured at the link layer in respect with the Advanced Encryption Standard (AES) scheme. In this case, a single hop security is provided. A basic example can be tested according to the CoapBlip application, already included inside the TinyOS environment ($TOSROOT/apps/CoapBlip). Two motes are needed: one server and one Point-to-Point Protocol (PPP) router ($TOSROOT/apps/PppRouter).

User can modify the list of available resources of the server mote inside the Makefile, such as DCOAP_RESOURCE_TEMP, DCOAP_RESOURCE_HUM, DCOAP_RESOURCE_VOLT and DCOAP_RESOURCE_LED. He can also change the IPv6 prefix DIN6_PREFIX.

```
COMPONENT=CoapBlipC
# uncomment this for network programming support
# BOOTLOADER=tosboot
#CFLAGS += -DCC2420_DEF_CHANNEL=21
#CFLAGS += -DRF230_DEF_CHANNEL=16
#CFLAGS += -DCC2420_DEF_RFPOWER=4
# Enable Low Power Listening
#CFLAGS += -DLOW_POWER_LISTENING
#CFLAGS += -DLPL_SLEEP_INTERVAL=512
# disables support for the AM stack, which somewhat reduces code size
# and compresses packet formats. If you want to use other tinyos
# protocols which are AM-based, you should not include this.
#CFLAGS += -DIEEE154FRAMES_ENABLED
# lib6lowpan contains inet_ntop6 and inet_pton6 to process ascii
# representations of IPv6 addresses. You can remove them to save some
# code if you don't use them
# CFLAGS += -DNO_LIB6LOWPAN_ASCII
# you can compile with or without a routing protocol... of course,
# without it, you will only be able to use link-local communication.
PFLAGS += -DRPL_ROUTING -DRPL_STORING_MODE -I$(LOWPAN_ROOT)/tos/lib/net/rpl
#PFLAGS += -DRPL_OF_MRHOF -DRPL_ROOT_ADDR=1
# if you set this, the stack won't use dhcp to assign addresses -- it
# will use this prefix combined with TOS_NODE_ID
PFLAGS += -DIN6_PREFIX=\"fec0::\"
# printf debugs. works only on telosb/tmote sky.
# CFLAGS += -DNEW_PRINTF_SEMANTICS -DPRINTFUART_ENABLED
# uncomment this line if you are having problems displaying the new printf semantics
# CFLAGS += -DPRINTF_BUFFER_SIZE=1024
################ CoAP Server ##############################
CFLAGS += -DCOAP_SERVER_ENABLED
# port where server is listening
CFLAGS += -DCOAP_SERVER_PORT=61616L
#set max uri length
```

```
CFLAGS += -DMAX_URI_LENGTH=5
# set available resources.
# Don't forget to change CFLAG: NUM_URIS!!!!!
CFLAGS += -DNUM_URIS=5
CFLAGS += -DCOAP_RESOURCE_TEMP
CFLAGS += -DCOAP_RESOURCE_HUM
CFLAGS += -DCOAP_RESOURCE_VOLT
#CFLAGS += -DCOAP_RESOURCE_ALL #TEMP + HUM + VOLT
CFLAGS += -DCOAP_RESOURCE_KEY
CFLAGS += -DCOAP_RESOURCE_LED
#CFLAGS += -DCOAP_RESOURCE_ROUTE
# timeout in milliseconds for sending PreACK
CFLAGS += -DCOAP_PREACK_TIMEOUT=500
#CFLAGS += -DINCLUDE_WELLKNOWN
################ CoAP Client ###################################
#CFLAGS += -DCOAP_CLIENT_ENABLED
# client destination port
CFLAGS += -DCOAP_CLIENT_PORT=61617L
# client IPv6 destination address
CFLAGS += -DCOAP_CLIENT_DEST=\"fec0::100\"
CFLAGS += -I.
include $(MAKERULES)
```

The mote is assumed connected to the port /dev/ttyUSB0. The motelist command permits the knowledge of where each sensor is connected (USB0, USB1, etc.). The compilation is done in respect with the following command, where X is the address:

```
make telosb blip coap install, X bsl,/dev/ttyUSB0
```

In the same way, the PPP router is set up with the following Makefile:

```
COMPONENT=PppRouterC
# PFLAGS += -DENABLE_SPI0_DMA
# PFLAGS += -DCC2420_DEF_CHANNEL=21
# use rpl
PFLAGS += -DRPL_ROUTING -DRPL_STORING_MODE -I$(LOWPAN_ROOT)/tos/lib/net/rpl
# PFLAGS += -DRPL_OF_MRHOF
# and ppp
PFLAGS += -I$(TOSDIR)/lib/ppp
PFLAGS += -I$(TOSDIR)/lib/fragpool
# this works around fragmentation in the RX buffer pool
PFLAGS += -DPPP_HDLC_RX_FRAME_LIMIT=1 -DPPP_HDLC_TX_FRAME_LIMIT=8
# if you set this, the stack won't use dhcp to assign addresses -- it
# will use this prefix combined with TOS_NODE_ID
PFLAGS += -DIN6_PREFIX=\"fec0::\"
# derive short address from the dhcp address assignment, if possible
#PFLAGS += -DBLIP_DERIVE_SHORTADDRS
include $(MAKERULES)
```

The PPP router is assumed to be connected to the port /dev/ttyUSB1:

```
make telosb blip install bsl, /dev/ttyUSB1
```

Then the User needs to start the driver of the Ppp Connection with the following command:

```
sudo pppd debug passive noauth nodetach 115200 /dev/ttyUSB1 nocrtscts nocdtrcts lcp-echo-interval 0 noccp noip ipv6 ::23,::24
```

Then the PPP connection is established with the command (the right IN6_PREFIX must be used):

```
sudo ifconfig ppp0 add fec0::100/64
```

The user then can request (GET/PUT) a resource (Voltage:sv, Temperature:st, Humidity:sh, Leds:l, AES key:ck) from the server mote with the coap-client application located in $TOSROOT/support/sdk/c/coap/examples:

```
./coap-client –m <CMD> coap://[fec0::X]:61616/<URI> -t binary
```

Here is the output for the st, st and sh resources:

```
\x42\x01\x01\x44\x11\x2A\x82\x73\x74
send to [fec0::2]:61616:
pdu (9 bytes) v:1 t:0 oc:2 c:1 id:324 o: 1:'*' 9:'st'
Jul 24 13:13:53 ** received from [fec0::2]:61616:
pdu (8 bytes) v:1 t:2 oc:1 c:80 id:324 o: 1:'*'
data:'\xF5w'
Jul 24 13:13:53 *** removed transaction 324
** process pdu: pdu (8 bytes) v:1 t:2 oc:1 c:80 id:324 o: 1:'*'
data:'\xF5w'
** Temperatur: 307.09 K
```

```
\x42\x01\x19\x55\x11\x2A\x82\x73\x76
send to [fec0::2]:61616:
pdu (9 bytes) v:1 t:0 oc:2 c:1 id:6485 o: 1:'*' 9:'sv'
Jul 24 13:16:42 ** received from [fec0::2]:61616:
pdu (8 bytes) v:1 t:2 oc:1 c:80 id:6485 o: 1:'*'
data:'\x1E\x01'
Jul 24 13:16:42 *** removed transaction 6485
** process pdu: pdu (8 bytes) v:1 t:2 oc:1 c:80 id:6485 o: 1:'*'
data:'\x1E\x01'
** Voltage: 2.86 V
```

```
\x42\x01\x47\x66\x11\x2A\x82\x73\x68
send to [fec0::2]:61616:
pdu (9 bytes) v:1 t:0 oc:2 c:1 id:18278 o: 1:'*' 9:'sh'
Jul 24 13:19:05 ** received from [fec0::2]:61616:
pdu (8 bytes) v:1 t:2 oc:1 c:80 id:18278 o: 1:'*'
data:'\xD2\x0D'
Jul 24 13:19:05 *** removed transaction 18278
** process pdu: pdu (8 bytes) v:1 t:2 oc:1 c:80 id:18278 o: 1:'*'
data:'\xD2\x0D'
** Humidity: 35.38 %
```

For instance, the following command permits to actuate on the LEDs of the server mote:

```
echo –e –n \\x02 ¦ ./coap-client –m put coap://[fec0::3]:61616/l –T 3a –t binary –f –
```

For a CM5000 mote, here is the matching table between LED values and colours:

- x00: no LED
- x01: Red
- x02: Yellow
- x03: Yellow and Red
- x04: Blue
- x05: Blue and Red
- x06: Blue and Yellow
- x07: Blue, Red and Yellow

Wireshark permits the capture of the data traffic of the ppp0 connection. When a resource request is launched, two packets are collected:

```
\x42\x01\xA8\x65\x11\x2A\x82\x73\x74
send to [fec0::2]:61616:
pdu (9 bytes) v:1 t:0 oc:2 c:1 id:43109 o: 1:'*' 9:'st'
Jul 24 13:36:13 ** received from [fec0::2]:61616:
pdu (8 bytes) v:1 t:2 oc:1 c:80 id:43109 o: 1:'*'
data:'\xF5w'
Jul 24 13:36:13 *** removed transaction 43109
** process pdu: pdu (8 bytes) v:1 t:2 oc:1 c:80 id:43109 o: 1:'*'
data:'\xF5w'
**.Temperatur: 307.09 K
```

First packet:

```
Linux cooked capture
        Packet type: Sent by us (4) 00 04
        Link-layer address type: 512 02 00
        Link-layer address length: 0 00 00
        Source: <Missing>00 00 00 00 00 00 00 00
        Protocol: IPv6 (0x86dd) 86 dd
Internet Protocol Version 6
        version:6 60
        Traffic class: 0x00000000 60 00 00 00
        FlowLabel: 0x00000000 60 00 00 00
        Payload length: 17 00 11
        Next header: UDP (0x11) 11
        Hop limit: 64 60
        Source: fec0::100 (fec0::100) fe c0 00 00 00 00 00 00 00 00 00 00 00 00 01 00
        Destination: fec0::2 (fec0::2) fe c0 00 00 00 00 00 00 00 00 00 00 00 00 00 02
User Datagram Protocol, Src Port : 61616 (61616), Dst Port : 61616 (61616)
        Source port: 61616 (61616) f0 b0
        Destination port: 61616 (61616) f0 b0
        Length : 17 00 11
```

Checksum : 0x2de2 [validation disabled] **2d e2**

Data (9 bytes)

Data: 4201A865112A827374 **42 01 a8 65 11 2a 82 73 74**

Length: 9

```
0000    00 04 02 00 00 00 00 00   00 00 00 00 00 00 86 dd   …….. ……..
0010    60 00 00 00 00 11 11 40   fe c0 00 00 00 00 00 00   `……@ ……..
0020    00 00 00 00 00 00 01 00   fe c0 00 00 00 00 00 00   …….. ……..
0030    00 00 00 00 00 00 00 02   f0 b0 f0 b0 00 11 2d e2   …….. ……-.
0040    42 01 a8 65 11 2a 82 73 74                          B..e.*.s t
```

Second Packet:

Linux cooked capture

Packet type: Sent by us (0) **00 00**

Link-layer address type: 512 **02 00**

Link-layer address length: 0 **00 00**

Source: <Missing>**00 00 00 00 00 00 00 00**

Protocol: IPv6 (0x86dd) **86 dd**

Internet Protocol Version 6

version:6 **60**

Traffic class: 0x00000000 **60 00 00 00**

FlowLabel: 0x00000000 **60 00 00 00**

Payload length: 16 **00 10**

Next header: UDP (0x11) **11**

Hop limit: 15 **0f**

Source: fec0::100 (fec0::100) **fe c0 00 00 00 00 00 00 00 00 00 00 00 00 01 00**

Destination: fec0::2 (fec0::2) **fe c0 00 00 00 00 00 00 00 00 00 00 00 00 00 02**

User Datagram Protocol, Src Port : 61616 (61616), Dst Port : 61616 (61616)

Source port: 61616 (61616) **f0 b0**

Destination port: 61616 (61616) **f0 b0**

Length : 16 **00 10**

Checksum : 0x0f91 [validation disabled] **0f 91**

Data (8 bytes)

Data: 6150A865112AF577 **61 50 a8 65 11 2a f5 77**

Length: 8

```
0000    00 00 02 00 00 00 00 00   00 00 00 00 00 00 86 dd   …….. ……..
0010    60 00 00 00 00 10 11 0f   fe c0 00 00 00 00 00 00   `……. ……..
0020    00 00 00 00 00 00 00 02   fe c0 00 00 00 00 00 00   …….. ……..
0030    00 00 00 00 00 00 01 00   f0 b0 f0 b0 00 10 0f 91   …….. ……..
0040    61 50 a8 65 11 2a f577                              aP.e.*.w
```

The user can set up a cryptographic key composed by 25 values used by the AES encryption. The key can be stored in a file. Its content can be updated with the PUT method:

```
Echo –e –n
\\xFF\\x01\\x02\\x03\\x04\\x05\\x06\\x07\\x08\\x01\\x02\\x03\\x04\\x05\\x06\\x07\\x08\\x09\\x10\\x11
\\x12\\x13\\x14\\x15\\x16>> key

./coap-client – m put coap://[fec0::2]:61616/ck –T 3a –t binary –f key
```

```
\x43\x03\xAD\x91\x11\x2A\x82\x63\x6B\x22\x33\x61\xFF\x01\x02\x03\x04\x05\x06\x07\x08\x01\x02\x03\x04\x05\x06\x07\x08\x09\x10\
x11\x12\x13\x14\x15\x16

send to [fec0::2]:61616:

pdu (37 bytes) v:1 t:0 oc:3 c:3 id:44433 o: 1:'*' 9:'ck' 11:'3a'

data:'\xFF\x01\x02\x03\x04\x05\x06\x07\x08\x01\x02\x03\x04\x05\x06\x07\x08\x09\x10\x11\x12\x13\x14\x15\x16'

Jul 24 15:17:06 ** retransmission #1 of transaction 44433

send to [fec0::2]:61616:

pdu (37 bytes) v:1 t:0 oc:3 c:3 id:44433 o: 1:'*' 9:'ck' 11:'3a'

data:'\xFF\x01\x02\x03\x04\x05\x06\x07\x08\x01\x02\x03\x04\x05\x06\x07\x08\x09\x10\x11\x12\x13\x14\x15\x16'

Jul 24 15:17:06 ** received from [fec0::2]:61616:

pdu (7 bytes) v:1 t:2 oc:1 c:80 id:44433 o: 11:'3a'

Jul 24 15:17:06 *** removed transaction 44433

** process pdu: pdu (7 bytes) v:1 t:2 oc:1 c:80 id:44433 o: 11:'3a'

** AES Key set
```

2 packets are emitted:

```
0000  00 04 02 00 00 00 00 00  00 00 00 00 00 00 86 dd   ........ ........
0010  60 00 00 00 00 2d 11 40  fe c0 00 00 00 00 00 00   `....-.@ ........
0020  00 00 00 00 00 00 01 00  fe c0 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 02  f0 b0 f0 b0 00 2d 89 a5   ........ .....-..
0040  43 03 ad 91 11 2a 82 63  6b 22 33 61 ff 01 02 03   C....*.c k"3a....
0050  04 05 06 07 08 01 02 03  04 05 06 07 08 09 10 11   ........ ........
0060  12 13 14 15 16                                     .....
```

```
0000  00 00 02 00 00 00 00 00  00 00 00 00 00 00 86 dd   ........ ........
0010  60 00 00 00 00 0f 11 0f  fe c0 00 00 00 00 00 00   `....... ........
0020  00 00 00 00 00 00 00 02  fe c0 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 01 00  f0 b0 f0 b0 00 0f fd d4   ........ ........
0040  61 50 ad 91 b2 33 61                               aP...3a
```

If we request the temperature resource again, we obtain:

```
\x42\x01\x9B\xEB\x11\x2A\x82\x73\x74
send to [fec0::2]:61616:
pdu (9 bytes) v:1 t:0 oc:2 c:1 id:39915 o: 1:'*' 9:'st'
Jul 24 15:17:25 ** received from [fec0::2]:61616:
pdu (8 bytes) v:1 t:2 oc:1 c:80 id:39915 o: 1:'*'
data:'\x09w'
Jul 24 15:17:25 *** removed transaction 39915
** process pdu: pdu (8 bytes) v:1 t:2 oc:1 c:80 id:39915 o: 1:'*'
data:'\x09w'
** Temperatur: 304.73 K
```

2 packets are also transmitted:

```
0000  00 04 02 00 00 00 00 00  00 00 00 00 00 00 86 dd   ........ ........
0010  60 00 00 00 00 11 11 40  fe c0 00 00 00 00 00 00   `......@ ........
0020  00 00 00 00 00 00 01 00  fe c0 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 02  f0 b0 f0 b0 00 11 3a 5c   ........ ......:\

0040  42 01 9b eb 11 2a 82 73  74                        B....*.s t
```

```
0000  00 00 02 00 00 00 00 00  00 00 00 00 00 00 86 dd   ........ ........
0010  60 00 00 00 00 10 11 0f  fe c0 00 00 00 00 00 00   `....... ........
0020  00 00 00 00 00 00 00 02  fe c0 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 01 00  f0 b0 f0 b0 00 10 08 0c   ........ ........
0040  61 50 9b eb 11 2a 09 77                           aP...*.w
```

3.1.6.2 CC2420 Standalone AES Encryption in TinyOS

The CC2420 chip is widely used in wireless sensor network module and other wireless transceivers. It provides hardware IEEE 802.15.4 MAC security operations, and standalone AES encryption that transforms each 128 bit plaintext into a 128 bit ciphertext. The standalone AES encryption enables the authentication between nodes and the design of block cipher-based hash functions. The main advantage between hardware and software AES encryption implementation relies on the speed of operations.

Bo Zhu et al. implemented the standalone hardware AES encryption for the MICAz module in the TinyOS 2.10 framework. They released the source code under BSD license, combined with an example.

- AesC.nc, AesP.nc, AesRamP.nc are the core function files
- Encrypt.nc is the interface file
- Install.sh is used for the installation
- Makefile is needed for the compilation
- TestAesAppC.nc and TestAesC.nc provide a simple test case

SplitControl and Encrypt interfaces are included in AesC.nc. The CC2420 chip is powered by SplitControl, before performing the AES encryption. Encrypt enables the encryption, based on three commands and two events. The command setKey(key) permits to store the 128 bits key into the specific CC2420 RAM area. After this storage completion, the event setKeyDone(key) will be reported. The command putPlain(plaintext, ciphertext) is used to copy the plaintext into the CC2420 RAM before the encryption. The ciphertext of the same size (128 bits) is then generated. After the encryption operation, the event getCipher(plaintext, ciphertext) will be notified. The user can modify the encryption key with the command clrKey(key).

Two types of security operations are provided, i.e. standalone encryption operation and inline security operation. The standalone encryption operates with a plaintext and keys of size (128 bits). A mote first writes the plaintext into the standalone buffer SABUF, and operates the encryption according to the SAES command. After the encryption completion, the ciphertext overwrites the plaintext in SABUF.

The inline security operation is more complex, but it supports encryption, decryption, and authentication. It is based on the reception buffer (RXFIFO) and the transmission buffer (TXFIFO) of CC2420. Three modes based on 128 bits AES keys are supported:

- Counter Mode Encryption (CTR): the outgoing MAC frames are encrypted in TXFIFO, and the incoming MAC frames are decrypted in RXFIFO
- Cipher Block Chaining Message Authentication Code (CBC-MAC) generate the Message Integrity Code (MIC) of messages
- Counter with CBC-MAC (CCM) combines CTR and CBC-MIC

CC2420 security features have been included in TinyOS 2.1.1 and later versions.

The CC2420 inline security presents two new interfaces, i.e. CC2420SecurityMode and CC2420Keys. Implementations (respectively interfaces) can be found in $TOSROOT/tos/chips/cc2420/security (respectively $TOSROOT/tos/chips/cc2420/interfaces)

The transmitter should be configured as what follows. Users can include the security features my adding the CC2420_HW_SECURITY flag in the application Makefile:

```
CFLAGS+=-DCC2420_HW_SECURITY
```

Here is the configuration file:

```
components new SecAMSenderC(AM_RADIO_COUNT_MSG) as AMSenderC;
components new AMReceiverC(AM_RADIO_COUNT_MSG);
components CC2420KeysC;
App.Receive -> AMReceiverC;
App.AMSend -> AMSenderC;
App.Packet -> AMSenderC;
App.CC2420SecurityMode -> AMSenderC;
App.CC2420Keys -> CC2420KeysC;
```

An array of 16 bytes is used to store the desired key values:

```
uint8_t key[16] = {0x98,0x67,0x7F,0xAF,0xD6,0xAD,0xB7,0x0C,0x59,0xE8,0xD9,0x47,0xC9,0x71,0x15,0x0F};
```

The receiver should be configured as what follows. Users can include the security features my adding the CC2420_HW_SECURITY flag in the application Makefile:

```
CFLAGS+=-DCC2420_HW_SECURITY
```

As a consequence, the decryption processes are enabled in CC2420ReceiveP.nc. Also, the receiver must know the key values used by a transmitter.

```
uint8_t key[16] = {0x98,0x67,0x7F,0xAF,0xD6,0xAD,0xB7,0x0C,0x59,0xE8,0xD9,0x47,0xC9,0x71,0x15,0x0F};
```

An example of the CC2420 inline security features (RadioCountToLeds and BaseStation) can be downloaded from http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x/apps/tests/cc2420/TestSecurity/ . It is already included in the TinyOS environment in $TOSROOT/apps/tests/cc2420/TestSecurity. Security features are added to outgoing packets. The packets are decrypted at the receiver node. User needs to compile one mote with ID 1 as the transmitter and other motes with any IDs to receive decrypted packets.

Here is the Makefile of the RadioCountToLeds application:

```
COMPONENT=RadioCountToLedsAppC
CFLAGS+=-DCC2420_HW_ACKNOWLEDGEMENTS
CFLAGS+=-DCC2420_HW_SECURITY
#CFLAGS+=-DCC2420_DEF_CHANNEL=25
CFLAGS+=-DTFRAMES_ENABLED
CFLAGS+=-DPACKET_LINK
CFLAGS+=-DTOSH_DATA_LENGTH=115
#CFLAGS+=-I%T/lib/printf
CFLAGS+=-I$(TOSDIR)/lib/printf
CFLAGS+=-DPRINTFUART_ENABLED
CFLAGS+=-I./
include $(MAKERULES)
```

```
make telosb blip install 1 bsl, /dev/ttyUSB0
make telosb blip install 2 bsl, /dev/ttyUSB1
```

The LEDS are programmed as follows:

- RED: a message is bridged from serial to radio

- GREEN: a message is bridge from radio to serial
- YELLOW/BLUE: a message is dropped due to queue overflow in either direction

User needs also to compile one mote with the BaseStation application.

```
make telosb blip install bsl, /dev/ttyUSB2
```

The BaseStation mote GREEN Led is normally blinking. User can launch the serial forwarder $TOSROOT/support/sdk/java/net/tinyos/sf/SerialForwarder. This application instantiates a server which provides a bi-directional packet stream between motes connected to the host PC and clients anywhere on the network.

```
java net.tinyos.sf.SerialForwarder -comm serial@/dev/ttyUSB2:telosb
```

3.1.6.3 TinySec in TinyOS

C Karlof et al. introduced the first fully implemented link-layer security suite for WSNs in 2004 [8]. In order to reduce the number of sent messages in a WSN where nodes are generally equipped with batteries, network processing such as aggregation and duplicate elimination is often used. As a consequence nodes in the network need to access and modify messages. Having security in higher layers would not allow such message manipulations. However end-to-end security can be preferred when nodes are planned to be connected to the Internet.

Two modes of operations are supported by TinySec, i.e. Authentication only (TinySec-Auth) and, Authenticated and Encryption (TinySec-AE). The data payload is encrypted using a Skipjack block cipher. Authentication is done based on CBC-MAC performed on the encrypted payload and the packet header. In TinySec-Auth, the entire packet is authenticated but the data is not encrypted. TinySec performs cryptography in software and does not take advantage of any hardware acceleration. The overall implementation of TinySec requires 256 bytes of RAM and 8152 bytes of ROM.

3.1.6.4 TinyECC2.0 in TinyOS

The development of Elliptic Curve Cryptography (ECC) has proven that Public Key Cryptography (PKC) can be used in WSNs. ECC is based on the algebraic structure of elliptic curves over finite fields. It was proposed independently by Victor Miller [9] and Neal Koblitz [10]. The harder it is to solve a mathematical problem, the more secure the algorithm.

For instance, the Rivest Shamir Adleman (RSA) algorithm is based on the Integer Factorization problem that has a sub-exponential solution. ECC is based on the Elliptic Curve Discrete Logarithmic Problem (ECDLP) where the solution is fully exponential. As a consequence, ECC can offer the same level of security than RSA with much smaller key sizes. A 160-bit ECC key provides the same level of security as a 1024-bit RSA key, and 224-bit ECC is equivalent to the 2048-bit RSA. Smaller keys result in faster computations, less memory, power and bandwidth consumption. Using ECC results in approximately 1.5 times smaller memory consumption on the motes and shows 4 to 5 times faster cryptography operations when compared to using RSA [11].

TinyECC is a portable and efficient library developed at the North Carolina State University. It has been released by A Liu et al. [11]. It provides a digital signature scheme, named Elliptic Curve Digital Signature Algorithm (ECDSA), a key exchange protocol named Elliptic Curve Diffie Hellman (ECDH) and a public key encryption scheme (ECIES). The last release of TinyECC2.0 supports different platforms such as MICA2/MICAz, TelosB/Tmote Sky, BSNV3 and Imote2 motes.

A user must install TinyOS 2.1.1 or a later version. He can also use a virtual machine where

the TinyOS is already set up, such as the one provided by the Advanticsys manufacturer.

He should extract TinyECC-2.0.zip to the following location:

```
sudo unzip TinyECC2.0.zip –d /opt/tinyos-2.x/apps/TinyECC-2.0.zip
```

Here is the list of interfaces provided by TinyECC2.0:

- NN.nc defines the interface NN implemented by NNM.nc, related to big natural number operations.
- ECC.nc defines the interface ECC implemented by ECCM.nc.It provides the basic and enhanced (sliding window method and projective coordinate system) elliptic curve operations.
- ECDSA.nc defines the interface ECDSA, implemented by ECDSAM.nc, which provides the ECDSA signature generation and verification.
- ECIES.nc defines the interface ECIES, implemented by ECIESM.nc. It performs the ECIES encryption and decryption.
- ECDH.nc defines the interface ECDH, implemented by ECDHM.nc, which computes the ECDH key establishment.
- SHA1.nc defines the interface SHA1, implemented by SHA1M.nc, which provides the SHA-1 functions.
- CurveParam.nc defines the interface CurveParam, which permits to get the parameters of elliptic curves and to optimize multiplication with omega. secp128*.nc, secp160*.nc, secp192*.nc implement this interface in order to provide parameters for Standards for Efficient Cryptography Group (SECG) defined elliptic curves. Curve name needs to be defined in the makefile to select the elliptic curve parameters.

TinyECC is written in NesC. TinyECC is a ready-to-use software package enabling ECC Public Key Cryptography operations. It includes optimizations for ECC operations. A. Liu et al. compared the execution time, ROM/RAM and energy consumptions for different platforms [11]. TinyECC supports ECC schemes such as ECDSA, ECDH and ECIES, defined in the Standards for Efficient Cryptography [12], and elliptic curve parameters such as secp160k1, secp160r1 and secp160r2 [13]. ECDH (respectively ECDSA) is a variant of the Diffie-Hellman key agreement protocol (respectively Digital Signature Algorithm). ECIES, also known as Elliptic Curve Augmented Encryption Scheme ECAES, supports semantic security. Optimizations for ECC include Barrett Reduction, Hybrid Multiplication, Hybrid Squaring, Projective Coordinate System, Sliding Window Method, Shamir's Trick and Curve-Specific Optimization [11]. TinyECC supports 128-bit, 160-bit and 192-bit ECC parameters. We remind that 160-bit ECC parameters provide the same security level as 1024-bit RSA. In their evaluation, A. Liu et al. selected secp160r1 to evaluate each optimization technique. They randomly generated the parameters and performed the average results on a set of 10 samples. Here are the conclusions.

For ECDSA, the execution time is ordered as what follows: PROJECTIVE > CURVE_OPT > HYBRID_MULT > HYBRID_SQR > SLIDING_WIN > SHAMIR_TRICK > BARRET. For the RAM (respectively ROM) they found SLIDING_WIN > SHAMIR_TRICK > BARRETT > HYBRID_MULT = HYBRID_SQR = CURV_OPT = PROJECTIVE (respectively PROJECTIVE > BARRETT ≈ SHAMIR_TRICK > CURV_OPT ≈ SLIDING_WIN > HYBRID_SQR > HYBRID_MULT).

They also provided the same analysis for ECIES and ECDH:

- ECIES:

- Execution time: PROJECTIVE > CURVE_OPT > HYBRID_MULT > HYBRID_SQR > SLIDING_WIN > BARRETT
- RAM: SLIDING_WIN > PROJECTIVE > BARRETT > HYBRID_MULT = HYBRID_SQR = CURVE_OPT
- ROM: PROJECTIVE ≈ BARRETT > CURVE_OPT > SLIDING_WIN > HYBRID_SQR > HYBRID_MULT (case A) and PROJECTIVE > SLIDING_WIN > BARRETT > HYBRID_SQR > HYBRID_MULT > CURVE_OPT (case B)

- ECDH:
  - Execution time: PROJECTIVE > CURVE_OPT > HYBRID_MULT > HYBRID_SQR > SLIDING_WIN > BARRETT
  - RAM: SLIDING_WIN > PROJECTIVE > BARRETT > HYBRID_MULT = HYBRID_SQR =CURVE_OPT
  - ROM: PROJECTIVE > BARRETT > SLIDING_WIN > HYBRID_SQR > HYBRID_MULT > CURVE_OPT (case A) and PROJECTIVE > SLIDING_WIN > BARRETT > HYBRID_SQR > HYBRID_MULT >CURVE_OPT

In the case A, all the other optimizations are disabled and then the consumption metrics are computed when the given optimization is enabled and disabled respectively. In the Case B, all the other optimizations are enabled and then the consumption metrics are computed when the given optimization is enabled and disabled respectively.

Enabling all optimizations requires long pre-computation and the largest ROM and RAM consumptions.

3.1.6.4.1 ECDSA

testECDSA.nc and testECDSAM.nc are related to ECDSA and permits to measure its execution time. As we are using TelosB/Tmote Sky mote, the installation is done as what follows. User must change the COMPONENT inside the Makefile of the folder TinyECC-2.0 with the correct value (in this case testECDSA).

```
# Makefile for testECDSA
COMPONENT=testECDSA
#change the packet length to 102 bytes
MSG_SIZE=102
#increase the task queue size
#PFLAGS += -DTOSH_MAX_TASKS_LOG2=8
##choose curve parameter
#CFLAGS+=-DSECP128R1
#CFLAGS+=-DSECP128R2
#CFLAGS+=-DSECP160K1
CFLAGS+=-DSECP160R1
#CFLAGS+=-DSECP160R2
#CFLAGS+=-DSECP192K1
#CFLAGS+=-DSECP192R1
#use test vector for secp160r1 to show the correctness of TinyECC
#CFLAGS+=-DTEST_VECTOR
#CFLAGS+=-DCODE_SIZE
##choose different optimization techniques
##NN
CFLAGS+=-DBARRETT_REDUCTION  #barrett reduction
```

```
CFLAGS+=-DHYBRID_MULT  #hybrid multipliation
CFLAGS+=-DHYBRID_SQR  #hybrid squre
CFLAGS+=-DCURVE_OPT  #optimization for secg curve
##ECC
CFLAGS+=-DPROJECTIVE  #projective coordinate
CFLAGS+=-DSLIDING_WIN  #sliding window method, window size is defined in ECC.h
##ECDSA
CFLAGS+=-DSHAMIR_TRICK  #shamir trick, windows size is defined in ECDSAM.nc
include $(MAKERULES)
```

It is possible to switch between seven curve parameters, i.e. DSECP128R1, DSECP128R2, DSECP160K1, DSECP160R1, DSECP160R2, DSECP192K1, DSECP192R1.

The application is then compiled and uploaded into a mote connected through a USB port.

```
make telosb install
```

The serial forwarder can be initialized with the following command:

```
java net.tinyos.sf.SerialForwarder -comm serial@/dev/ttyUSB0:telosb &
```

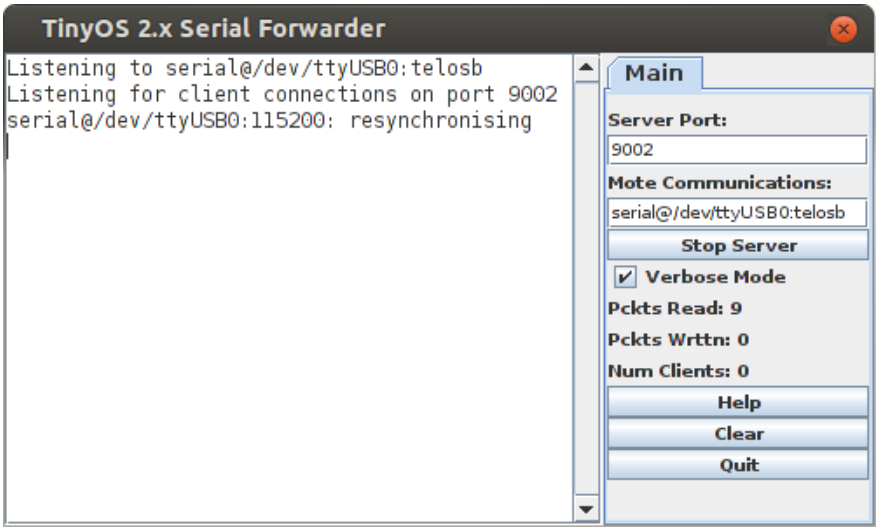

Figure 8: TinyOS Serial Forwarder

The user can start the application by typing the following command from a shell:

```
cd /opt/tiny/opt/tinyos-2.1.1/apps/TineECC-2.0/
java show_ecdsa
```

The output is directly displayed on the console. Many rounds are generated and provide the following information:

- The private key **d**
- The time of ECC.init() **ecci**
- The public key **x** and **y**
- Time of public key generation **pkg**
- Time of ECDSA.init() **ecdsai**
- The message **msg**
- The message signature **r** and **s**
- The time of signature generation **sg**

- The time of signature verification **sv**

Average timing result is also computed after each new round.

```
---------- round 1 ----------
Private key:
d: 58a9a30d5937a46a5a8d5780483bf86e533ad08
[ time of ECC.init() is 2.462 sec ]
Public key:
x: b23aecbc4a85845975f1c9dd3776aa07d0e0d728
y: 97544c3f505ce484beb29c851432abf85efcb138
[ time of public key generation is 2.631 sec ]
[ time of ECDSA.init() is 4.76 sec ]
content and signature
msg: bd65d5b57ce2dba944973174f7f1f4f7f8efc19d2c42963e63d9ad459c2751b472fbe0dfa85aa653b964d7b87aebc98d0c021b20
signature
r: fc0b8add4d9a036e18c04cd872cd3297bf4b1ed6
s: 0220bb36907191b99db37c53721b6343e8495535
[ time of signature generation is 2.832 sec ]
[ time of signature verification is 3.665 sec ] (pass)
Average timing result
ECC.init(): 2.4635
ECDSA.init(): 4.754
public key gen: 2.6605
sign: 2.8705
verify: 5.491
```

We ran several experiments with the available curve parameters.

ECDSA DSECP128R1	**ecci**	**pkg**	**ecdsai**	**sg**	**sv**
1	1.795	2.733	3.448	2.91	3.712
2	1.795	2.817	3.458	2.863	3.626
3	1.795	2.805	3.458	2.898	3.719
4	1.796	2.79	3.459	2.829	3.703
5	1.795	2.789	3.454	2.922	3.64
6	1.795	2.8	3.462	2.857	3.642
7	1.794	2.786	3.472	2.901	3.73
8	1.795	2.82	3.465	2.846	3.648
9	1.794	2.78	3.451	2.9	3.732
10	1.793	2.823	3.448	2.814	3.693

Table 4: Experimentation of TinyECC with ECDSA DSECP128R1

ECDSA DSECP128R2	**ecci**	**pkg**	**ecdsai**	**sg**	**sv**
1	1.789	3.405	3.448	3.052	4.088
2	1.789	3.02	3.445	3.154	4.079
3	1.789	3.04	3.435	3.087	3.964
4	1.789	2.924	3.458	3.131	3.939
5	1.789	3.054	3.463	3.074	4.073
6	1.789	2.914	3.437	3.037	4.071

7	1.794	3.065	3.463	3.043	3.987
8	1.795	2.976	3.46	3.175	4.044
9	1.795	3.072	3.474	3.099	4.03
10	1.795	3.046	3.47	3.144	4.134

Table 5: Experimentation of TinyECC with ECDSA DSECP128R2

ECDSA DSECP160R1	**ecci**	**pkg**	**ecdsai**	**sg**	**sv**
1	2.457	2.682	4.734	2.9	3.613
2	2.453	2.623	4.745	2.823	3.653
3	2.457	2.663	4.727	2.897	3.689
4	2.461	2.685	4.753	2.859	3.641
5	2.465	2.737	4.751	2.877	3.676
6	2.463	2.698	4.762	2.887	3.664
7	2.466	2.642	4.788	2.888	3.702
8	2.463	2.688	4.745	2.893	3.702
9	2.466	2.721	4.758	2.86	3.685
10	2.462	2.716	4.76	2.915	3.692

Table 6: Experimentation of TinyECC with ECDSA DSECP160R1

ECDSA DSECP160R2	**ecci**	**pkg**	**ecdsai**	**sg**	**sv**
1	2.438	2.817	4.721	2.949	3.768
2	2.435	2.792	4.728	2.972	3.754
3	2.439	2.758	4.736	2.951	3.824
4	2.435	2.781	4.744	2.966	3.706
5	2.439	2.797	4.737	2.898	3.82
6	2.435	2.8	4.698	2.976	3.785
7	2.439	2.771	4.717	2.951	3.76
8	2.436	2.73	4.714	2.895	3.76
9	2.44	2.745	4.739	2.988	3.72
10	2.428	2.768	4.724	2.91	3.767

Table 7: Experimentation of TinyECC with ECDSA DSECP160R2

ECDSA DSECP192K1	ecci	pkg	ecdsai	sg	sv
1	3.409	4.16	6.59	4.324	5.171
2	3.405	4.08	6.568	4.368	5.273
3	3.41	3.99	6.583	4.419	5.213
4	3.406	4.15	6.595	4.403	5.171
5	3.411	4.163	6.571	4.409	5.212
6	3.406	4.146	6.572	4.383	5.141
7	3.411	4.15	6.574	4.416	5.245
8	3.407	4.117	6.563	4.359	5.183
9	3.412	4.126	6.58	4.413	5.247
10	3.406	4.172	6.549	4.321	5.152

Table 8: Experimentation of TinyECC with ECDSA DSECP192K1

ECDSA DSECP192R1	ecci	pkg	ecdsai	sg	sv
1	3.421	4.101	6.6	4.337	5.421
2	3.417	4.107	6.582	4.363	5.511
3	3.421	4.104	6.581	4.395	5.549
4	4.416	4.069	5.568	4.302	5.371
5	3.422	4.075	6.618	4.357	5.55
6	3.414	4.113	6.59	4.26	5.5
7	3.421	4.105	6.592	4.276	5.461
8	3.416	4.036	6.566	4.331	5.449
9	3.421	3.931	6.62	4.377	5.425
10	3.416	4.106	6.58	4.354	5.434

Table 9: Experimentation of TinyECC with ECDSA DSECP192R1

3.1.6.5 ECIES

testECIES.nc and testECIESM.nc are related to ECIES and permits the measurement of its execution time. As we are using TelosB/Tmote Sky mote, the installation is done per what follows. User must change the COMPONENT inside the Makefile of the folder TinyECC-2.0 with the correct value (in this case testECIES).

```
#makefile for testECIESCOMPONENT=testECIES
#change the packet length to 102 bytes
MSG_SIZE=102
#increase the queue size to 256
#PFLAGS += -DTOSH_MAX_TASKS_LOG2=6
##choose curve parameter
#CFLAGS+=-DSECP128R1
#CFLAGS+=-DSECP128R2
#CFLAGS+=-DSECP160K1
CFLAGS+=-DSECP160R1
#CFLAGS+=-DSECP160R2
#CFLAGS+=-DSECP192K1
#CFLAGS+=-DSECP192R1
#use test vector for secp160r1 to show the correctness of TinyECC
#CFLAGS+=-DTEST_VECTOR
#CFLAGS+=-DCODE_SIZE
##choose different optimization techniques
```

```
##NN
CFLAGS+=-DBARRETT_REDUCTION  #barrett reduction
CFLAGS+=-DHYBRID_MULT  #hybrid multipliation
CFLAGS+=-DHYBRID_SQR  #hybrid squre
CFLAGS+=-DCURVE_OPT  #optimization for secg curve
##ECC
CFLAGS+=-DPROJECTIVE  #projective coordinate
CFLAGS+=-DSLIDING_WIN  #sliding window method, windows size is defined in ECC.h
include $(MAKERULES)
```

It is possible to switch between seven curve parameters, i.e. DSECP128R1, DSECP128R2, DSECP160K1, DSECP160R1, DSECP160R2, DSECP192K1, DSECP192R1.

The application is then compiled and uploaded into a mote connected through a USB port.

```
make telosb install
```

The serial forwarder can be initialized with the following command:

```
java net.tinyos.sf.SerialForwarder -comm serial@/dev/ttyUSB0:telosb &
```

The user can start the application by typing the following command from a shell:

```
cd /opt/tiny/opt/tinyos-2.1.1/apps/TineECC-2.0/
java show_ecies
```

The output is directly displayed on the console. Many rounds are generated and provide the following information:

- The plaintext message **m**
- The time of ECIES.init() **eciesi**
- The private key **d**
- The public key **x** and **y**
- Time of public key generation **pkg**
- Time of ECIES.encrypt **eciese**
- The ciphertext message **c**
- Time of ECIES.decrypt **eciesd**

Average timing result is also computed after each new round.

```
6bc0aa4b890d0c02163379edcc821b2944923b69cd851c2751bd65d5b57ce2dba944973174f7f1f4
[ time of ECIES.init() is 2.471 sec ]
Private key:
d: 3bd90d63163e9396d140fb63e775b1ed5679dea1
Public key:
x: e727d37663ef9b77fdfeba4ebee979d0c6b89ad
y: f2735e2aee45d84fd2bada7360de6a174f1bc2ca
[ time of public key generation is 2.719 sec ]
0210da4922e2993ea55344a7321c42eacd11a01e107332b4dbc397ddf9191941e46bfa6745d46f7f3c69660cd6eb1c728f14d59b48c6751b882ab
5e344565024eef4ed0dd7b80ae025b4854c72b687b471
[ time of ECIES.encrypt() is 6.005 sec ]
6bc0aa4b890d0c02163379edcc821b2944923b69cd851c2751bd65d5b57ce2dba944973174f7f1f4
```

[time of ECIES.decrypt() is 3.945 sec]

Average timing result for 1 rounds

ECIES.init(): 2.471

public key gen: 2.719

encrypt: 11.919001

decrypt: 7.8929996

ECIES DSECP128R1	eciesi	pkg	eciese	eciesd
1	1.801	2.792	6.17	3.722
2	1.801	2.809	6.236	3.746
3	1.802	2.808	6.226	3.761
4	1.802	2.773	6.206	3.684
5	1.803	2.741	5.739	3.674
6	1.802	2.826	6.159	3.78
7	1.802	2.805	6.156	3.769
8	1.803	2.804	6.239	3.766
9	1.802	2.811	6.194	3.764
10	1.803	2.764	6.064	3.687

Table 10: Experimentation of TinyECC with ECIES DSECP128R1

ECIES DSECP128R2	eciesi	pkg	eciese	eciesd
1	1.792	2.99	6.67	3.91
2	1.791	2.967	6.172	3.889
3	1.791	2.953	6.589	3.909
4	1.792	2.935	6.607	3.88
5	1.792	3.03	6.689	3.996
6	1.792	3.033	6.446	3.977
7	1.792	2.893	6.506	3.805
8	1.798	3.021	6.637	3.967
9	1.799	3.044	6.523	3.993
10	1.799	3.045	6.704	4.003

Table 11: Experimentation of TinyECC with ECIES DSECP128R2

ECIES DSECP160R1	eciesi	pkg	eciese	eciesd
1	2.455	2.703	5.97	3.95
2	2.455	2.686	5.964	3.916
3	2.455	2.667	5.91	3.885
4	2.455	2.619	5.98	3.804
5	2.455	2.698	5.877	3.924
6	2.455	2.703	5.967	3.921
7	2.455	2.663	5.856	3.891
8	2.456	2.727	5.817	3.947
9	2.456	2.688	5.917	3.886
10	2.457	2.687	5.942	3.898

Table 12: Experimentation of TinyECC with ECIES DSECP160R1

ECIES DSECP160R2	eciesi	pkg	eciese	eciesd
1	2.433	2.804	5.982	4.084
2	2.433	2.78	6.092	4.061

3	2.433	2.771	6.1	4.034
4	2.432	2.748	6.135	4.038
5	2.432	2.752	6.137	4.019
6	2.433	2.75	6.079	4.019
7	2.433	2.765	6.14	4.025
8	2.433	2.786	6.044	4.044
9	2.433	2.788	6.141	4.054
10	2.441	2.783	6.081	4.085

Table 13: Experimentation of TinyECC with ECIES DSECP160R2

ECIES DSECP192K1	**eciesi**	**pkg**	**eciese**	**eciesd**
1	3.42	4.16	9.046	6.083
2	3.42	4.184	9.015	6.117
3	3.42	4.179	9.052	6.13
4	3.421	4.098	9.106	6.011
5	3.42	4.165	9.03	6.088
6	3.421	4.147	8.995	6.077
7	3.421	4.137	8.937	6.079
8	3.394	4.08	9.031	5.965
9	3.395	4.06	8.826	5.981
10	3.396	4.142	8.986	6.073

Table 14: Experimentation of TinyECC with ECIES DSECP192K1

ECIES DSECP192R1	**eciesi**	**pkg**	**eciese**	**eciesd**
1	3.403	4.1	8.883	5.878
2	3.403	4.074	8.856	5.839
3	3.404	4.092	8.845	5.892
4	3.404	4.089	8.861	5.907
5	3.404	4.022	8.927	5.784
6	3.405	4.075	8.872	5.867
7	3.404	4.059	8.808	5.846
8	3.405	4.053	8.779	5.844
9	3.404	4.036	8.928	5.789
10	3.405	4.014	8.703	5.803

Table 15: Experimentation of TinyECC with ECIES DSECP192R1

3.1.6.6 ECDH

testECDH.nc and testECDHM.nc are related to ECDH and permits the measurement of its execution time. As we are using TelosB/Tmote Sky mote, the installation is done per what follows. User must change the COMPONENT inside the Makefile of the folder TinyECC-2.0 with the correct value (in this case testECDH).

```
#makefile for testECDH
COMPONENT=testECDH
#change the packet length to 102 bytes
MSG_SIZE=102
#increase task queue size to 256
#PFLAGS += -DTOSH_MAX_TASKS_LOG2=8
##choose curve parameter
#CFLAGS+=-DSECP128R1
#CFLAGS+=-DSECP128R2
#CFLAGS+=-DSECP160K1
```

```
CFLAGS+=-DSECP160R1
#CFLAGS+=-DSECP160R2
#CFLAGS+=-DSECP192K1
#CFLAGS+=-DSECP192R1
#use test vector for secp160r1 to show the correctness of TinyECC
#CFLAGS+=-DTEST_VECTOR
#CFLAGS+=-DCODE_SIZE
##choose different optimization techniques
##NN
CFLAGS+=-DBARRETT_REDUCTION  #barrett reduction
CFLAGS+=-DHYBRID_MULT  #hybrid multipliation
CFLAGS+=-DHYBRID_SQR  #hybrid squre
CFLAGS+=-DCURVE_OPT  #optimization for secg curve
##ECC
CFLAGS+=-DPROJECTIVE  #projective coordinate
CFLAGS+=-DSLIDING_WIN  #sliding window method
include $(MAKERULES)
```

It is possible to switch between seven curve parameters, i.e. DSECP128R1, DSECP128R2, DSECP160K1, DSECP160R1, DSECP160R2, DSECP192K1, DSECP192R1.

The application is then compiled and uploaded into a mote connected through a USB port.

```
make telosb install
```

The serial forwarder can be initialized with the following command:

```
java net.tinyos.sf.SerialForwarder -comm serial@/dev/ttyUSB0:telosb &
```

The user can start the application by typing the following command from a shell:

```
cd /opt/tiny/opt/tinyos-2.1.1/apps/TineECC-2.0/
java show_ecdh
```

The output is directly displayed on the console. Many rounds are generated and provide the following information:

- The time of ECDH.init **ecdhi**
- The private key1 **d1**
- The public key1 **x1** and **y1**
- The time of public key1 generation **pkg1**
- The private key2 **d2**
- The public key2 **x2** and **y2**
- The time of public key2 generation **pkg2**
- The established key1 **ek1**
- Time of ECDH.key_agree(1) **ecdhka1**
- The established key2 **ek2**
- Time of ECDH.key_agree(2) **ecdhka2**

```
[ time of EDH.init() is 2.462 sec ]
Private key1:
```

76f12bf705749a315d95497b8e81a4876ead61c5

Public key1:

x: f6c6d38995b21ef32826c30fe94bd2977fc85d2

y: a2bf24aaf788a969724cdf5436e5a56b1e652423

[time of public key 1 generation is 2.74 sec]

Private key2:

53ad39590c23169e93c4dca1f6f931c7964d8695

Public key2:

x: 4c84d0d86fdc064dfea10cbac47d833b7f149809

y: dd7537d9dda79514c40948c7dcb8b7acf0cb0a15

[time of public key 2 generation is 2.731 sec]

established key1: 011294f5a4dbdbda23380c6e9ec1e48e84356200

[time of ECDH.key_agree() for 1 is 3.198 sec]

established key2: 011294f5a4dbdbda23380c6e9ec1e48e84356200

[time of ECDH.key_agree() for 2 is 3.21 sec]

Average timing result for 1 rounds

ECDH.init(): 2.462

PK1: 2.74

PK2: 2.731

key_agree1: 3.198

key_agree2: 3.21

ECDH SECP128R1	**ecdhi**	**pkg1**	**pkg2**	**ecdhka1**	**ecdhka2**
1	1.794	2.773	2.797	3.337	3.378
2	1.795	2.756	2.735	3.313	3.277
3	1.795	2.794	2.789	3.378	3.379
4	1.794	2.804	2.798	3.366	3.353
5	1.795	2.799	2.726	3.371	3.285
6	1.795	2.784	2.75	3.383	3.301
7	1.795	2.745	2.79	3.321	3.355
8	1.799	2.782	2.805	3.325	3.367
9	1.8	2.819	2.782	3.406	3.362
10	1.8	2.787	2.784	3.35	3.355

Table 16: Experimentation of TinyECC with ECHD SECP128R1

ECDH SECP128R2	**ecdhi**	**pkg1**	**pkg2**	**ecdhka1**	**ecdhka2**
1	1.796	2.921	3.048	3.469	3.627
2	1.796	2.901	2.978	3.459	3.531
3	1.795	2.951	3.034	3.521	3.623
4	1.796	3.046	3.053	3.615	3.604
5	1.796	3.035	2.97	3.608	3.521
6	1.796	3.05	3.005	3.618	3.535
7	1.796	3.0	3.034	3.559	3.607
8	1.795	3.031	2.942	3.569	3.492
9	1.796	3.067	3.019	3.644	3.59
10	1.796	3.015	3.025	3.571	3.583

Table 17: Experimentation of TinyECC with ECHD SECP128R2

ECDH SECP160R1	ecdhi	pkg1	pkg2	ecdhka1	ecdhka2
1	2.462	2.74	2.731	3.198	3.21
2	2.463	2.712	2.697	3.196	3.171
3	2.462	2.7	2.681	3.164	3.166
4	2.463	2.702	2.748	3.169	3.257
5	2.463	2.705	2.641	3.167	3.132
6	2.462	2.716	2.712	3.18	3.199
7	2.463	2.726	2.685	3.218	3.173
8	2.463	2.699	2.681	3.179	3.147
9	2.463	2.674	2.714	3.145	3.196
10	2.463	2.732	2.713	3.21	3.196

Table 18: Experimentation of TinyECC with ECHD SECP160R1

ECDH SECP160R2	ecdhi	pkg1	pkg2	ecdhka1	ecdhka2
1	2.434	2.783	2.821	3.271	3.335
2	2.435	2.781	2.785	3.277	3.302
3	2.435	2.77	2.796	3.251	3.325
4	2.435	2.749	2.786	3.234	3.29
5	2.435	2.8	2.786	3.3	3.285
6	2.435	2.751	2.767	3.23	3.259
7	2.436	2.762	2.77	3.257	3.255
8	2.436	2.796	2.735	3.316	3.226
9	2.436	2.741	2.813	3.191	3.344
10	2.436	2.771	2.721	3.263	3.199

Table 19: Experimentation of TinyECC with ECHD SECP160R2

ECDH SECP192K1	ecdhi	pkg1	pkg2	ecdhka1	ecdhka2
1	3.412	4.121	4.124	4.805	4.811
2	3.411	4.14	4.115	4.851	4.783
3	3.412	4.09	4.158	4.749	4.891
4	3.412	4.15	4.171	4.827	4.869
5	3.411	4.083	4.125	4.756	4.784
6	3.411	4.109	4.145	4.779	4.825
7	3.411	4.163	4.141	4.888	4.841
8	3.411	4.104	4.079	4.799	4.748
9	3.411	4.137	4.117	4.829	4.815
10	3.398	4.139	4.153	4.814	4.845

Table 20: Experimentation of TinyECC with ECHD SECP192K1

ECDH SECP192R1	ecdhi	pkg1	pkg2	ecdhka1	ecdhka2
1	3.407	4.061	4.065	4.731	4.721
2	3.407	4.088	4.041	4.764	4.711
3	3.407	4.024	4.099	4.673	4.809
4	3.407	4.077	4.103	4.753	4.774
5	3.407	4.021	4.062	4.671	4.696
6	3.407	4.044	4.064	4.699	4.757
7	3.406	4.104	4.092	4.808	4.76

8	3.406	4.039	4.019	4.715	4.66
9	3.406	4.079	4.066	4.75	4.726
10	3.405	4.083	4.096	4.741	4.775

Table 21: Experimentation of TinyECC with ECHD SECP192R1

3.1.6.7 ContikiSec in Contiki

In 2009, L. Casado et al introduced in [14] a secure network layer for wireless sensor networks under the Contiki OS. It was named ContikiSec. It provides three modes of operation, i.e. confidentiality only (ContikiSec-Enc), authentication only (ContikiSec-Auth) and both authentication with encryption (ContikiSec-AE).

After evaluation of six different block ciphers (AES, RC5, Skipjack, Triple-DES, Twofish and Extended Tiny Encryption Algorithm XTEA). AES-CBC-CS (Cipher Block Chaining-Ciphertext Stealing) has been selected for the encryption only mode. This symmetric block cipher requires an initialization vector (IV) consisting of a random block of data. Contiki-Auth uses CMAC algorithm (Cipher-based Message Authentication Code) for generating a 4-byte MAC. Contiki-AE that achieves the highest level of security by providing confidentiality, authentication and integrity, uses Offset Codebook Mode (OCB) which is based on AES.

The authors tested ContikiSec on the MSB-430 platform. The compilation of the source code with and without the security primitives permits to retrieve the needed ROM space. The RAM memory is estimated with the msp430-ram-usage tool. The energy consumption can be evaluated with the software-based on-line energy estimation mechanism Energest. Authors showed the time needed to encrypt and decrypt a single block of plaintext (64-bit for XTEA, Skipjack and Triple-DES, and 128-bit for AES and Twofish). They showed that Skipjack (80-bit key) is the fastest cipher, but its key expansion process is the slowest one. Skipjack provides the largest throughput. AES achieves reasonable memory usage and energy consumption.

One major disadvantage of ContikiSec is the usage of the simplest key management mechanism where all nodes in the network present the same 128-bit key. The whole network is much more vulnerable to various attacks. In fact, if only one node is compromised, then the entire network becomes unsecure.

3.1.6.8 IPSec in Contiki

IPSec (Internet Protocol Security) operates in the network layer. It is based on two protocols, i.e. the Authentication Header (AH) and the Encapsulating Security Protocol (ESP). AH provides integrity and data origin authentication with optional anti-reply features while ESP also offers confidentiality.

Two modes are available, i.e. transport and tunnel. In the transport mode, the IPSec header is inserted after the IP header. Protection is primarily provided for next layer protocols. In fact only the payload of the original IP packet is encrypted and/or authenticated. In the tunnel mode, a complete IP packet is encrypted and/or authenticated. Then it is encapsulated into a new IP packet with a new IP header.

A Security Association (SA) is an association between two IPSec endpoints. The flow of information in one direction is protected. As a consequence, two SAs are needed for a bidirectional communication. SAs can be established and maintained manually. However the Internet Key Exchange protocol (IKE or IKEv2) is used to automatically do this. Mutual authentication is performed between two parties. The IKE security association is used to efficiently establish SAs for ESP or AH.

All IKE communications consist of pairs of messages, called “exchanges”. The IKE SA INIT

exchange negotiates security parameters (nonce and Diffie-Hellman) for the IKE SA. The IKE AUTH exchange transmits identities, authenticates them and initializes the SA. IKE is responsible for key management needed in all encryption and authentication operations based on keys.

IPSec for 6LoWPAN has been released by Dr Shahid Raza from the Swedish Institute of Computer Science (SICS) [15]. The source code is available here:

```
svn co https://contikiprojects.svn.sourceforge.net/svnroot/contikiprojects/sics.se/ipsec ipsec
```

The set-up is based on 2 motes connected through USB ports to a computer running Instant contiki 2.6.

The first mote M1 is used as a border router.

```
M1: MAC 00:12:74:00:10:21:b4:d8
aaaa::212:7400:1021:b4d8
```

The second mote is used as Application Example.

```
M2: MAC 00:12:74:00:10:f4:10:3a
aaaa::212:7400:10f4:103a
```

IPSec is first configured inside the Ubuntu virtual machine (Instant contiki). The package **ipsec-tools** must be installed.

```
sudo apt-get install ipsec-tools
```

The file ipsec-tools.conf must be edited in order to specify SAs:

```
sudo gedit /etc/ipsec-tools.conf
```

The IPv6 address of the Mote 2 (Application Example) is **aaaa::212:7400:10f4:103a**.

In order to start the service, the following command should be run:

```
sudo chmod 750 ipsec-tools.conf #(conf file not readable to the world)
sudo /etc/init.d/setkey start
```

```
user@instant-contiki:/etc$ sudo /etc/init.d/setkey start
* Loading IPSec SA/SP database:
[ OK ]
```

For the part related to the configuration of IPSec in the Contiki mote, it is mentioned that, the user should set in **project.conf** the selected modes of operation:

```
/*
* Copyright (c) 2010, Swedish Institute of Computer Science.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the above copyright
* notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
* notice, this list of conditions and the following disclaimer in the
```

```
* documentation and/or other materials provided with the distribution.
* 3. Neither the name of the Institute nor the names of its contributors
* may be used to endorse or promote products derived from this software
* without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ``AS IS'' AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*
* $Id: project-conf.h,v 1.1 2011/09/28 13:11:08 simonduq Exp $
*/
#ifndef __PROJECT_H__
#define __PROJECT_H__
#ifndef UIP_FALLBACK_INTERFACE
#define UIP_FALLBACK_INTERFACE rpl_interface
#endif
/* Save some memory */
#define UIP_CONF_TCP 0
#undef UIP_CONF_DS6_NBR_NBU
#define UIP_CONF_DS6_NBR_NBU 12
#undef UIP_CONF_DS6_ROUTE_NBU
#define UIP_CONF_DS6_ROUTE_NBU 12
/* Tmote Sky's serial works better with DMA */
#define UART1_CONF_RX_WITH_DMA 1
/* AH and ESP can be enabled/disabled independently */
#define WITH_CONF_IPSEC_AH 1
#define WITH_CONF_IPSEC_ESP 1
/* Configuring an AES implementation */
#define CRYPTO_CONF_AES cc2420_aes
/* Configuring a cipher block mode of operation (encryption/decryption) */
#define IPSEC_CONF_BLOCK aesctr
/* Configuring a cipher block MAC mode of operation (authentication) */
#define IPSEC_CONF_MAC aesxcbc_mac
#undef RF_CHANNEL
#define RF_CHANNEL 26
#undef NETSTACK_CONF_RDC
#define NETSTACK_CONF_RDC contikimac_driver
//#define NETSTACK_CONF_RDC nullrdc_driver
/* Enable nullrdc autoack handling */
```

```
//#undef NULLRDC_CONF_802154_AUTOACK
//#define NULLRDC_CONF_802154_AUTOACK 1
#endif /* __PROJECT_H__ */
```

In this case, IPSEC AH and ESP will be used.

The key are stored in ipsec/aes-ctr.c and ipsec/aes-xcbc-mac.c

```
/**
* \file
* AES-CTR block cipher mode of operation
* \author
* Simon Duquennoy <simonduq@sics.se>
*/
#include <stdlib.h>
#include "net/uip.h"
#include "ipsec.h"
#include "aes-moo.h"
/*---------------------------------------------------------------------------*/
/* pre-shared AES key */
static const unsigned char aes_key[IPSEC_KEYSIZE] =
{0xcf,0x5f,0xaa,0xca,0x70,0xee,0x5e,0xc4,0xc8,0xf4,0x31,0x58,0xa4,0x5c,0x03,0x63};
/* pre-shared AES-CTR nonce */
static const unsigned char aes_ctr_nonce[] =
{0x69,0xbb,0xc0,0xc9};
#define AESCTR_NONCESIZE 4
/*---------------------------------------------------------------------------*/
struct aes_ctr {
u8_t counter[IPSEC_KEYSIZE];
} aes_ctr;
/*---------------------------------------------------------------------------*/
static void
aes_ctr_init(struct aes_ctr *actr, const unsigned char *key,
const unsigned char *iv, const unsigned char *nonce)
{
/* Set key */
CRYPTO_AES.init(key);
/* Initialize counter block */
memcpy(actr->counter, nonce, AESCTR_NONCESIZE);
memcpy(actr->counter + AESCTR_NONCESIZE, iv, IPSEC_IVSIZE);
*((u32_t *)(actr->counter + AESCTR_NONCESIZE + IPSEC_IVSIZE)) = UIP_HTONL(1);
}
/*---------------------------------------------------------------------------*/
static void
aes_ctr_step(struct aes_ctr *actr, char *buff, int len)
{
int i;
char tmp[IPSEC_KEYSIZE];
u32_t count;
if(len == 0) {
```

```
return;
}
/* tmp = counter */
memcpy(tmp, actr->counter, IPSEC_KEYSIZE);
/* AES encrypt tmp */
CRYPTO_AES.encrypt(tmp);
/* buff ^= tmp */
for (i=0; i<len; i++) {
buff[i] ^= tmp[i];
}
/* counter++ */
count = UIP_HTONL(*((uint32_t*)(actr->counter + 12)));
*((uint32_t*)(actr->counter + 12)) = UIP_HTONL(count+1);
}
/*---------------------------------------------------------------------------*/
static void
aes_ctr_process(unsigned char *buff, int bufflen,
const unsigned char *iv, const unsigned char *key, const unsigned char *nonce)
{
int i;
struct aes_ctr actr;
aes_ctr_init(&actr, key, iv, nonce);
for(i=0; i<bufflen/IPSEC_KEYSIZE; i++) {
aes_ctr_step(&actr, buff + i * IPSEC_KEYSIZE, IPSEC_KEYSIZE);
}
aes_ctr_step(&actr, buff + i * IPSEC_KEYSIZE, bufflen % IPSEC_KEYSIZE);
}
/*---------------------------------------------------------------------------*/
void
encrypt_decrypt(unsigned char *buff, int bufflen, const unsigned char *iv)
{
aes_ctr_process(buff, bufflen, iv, aes_key, aes_ctr_nonce);
}
/*---------------------------------------------------------------------------*/
struct ipsec_encrypt_implem aesctr = {
encrypt_decrypt,
encrypt_decrypt,
};
/*---------------------------------------------------------------------------*/
```

The key is the same than the one used during the IPSec configuration in Ubuntu (0xcf5faaca70ee5ec4c8f43158a45c0363):

```
{0xcf,0x5f,0xaa,0xca,0x70,0xee,0x5e,0xc4,0xc8,0xf4,0x31,0x58,0xa4,0x5c,0x03,0x63}
```

The nonce is {0x69,0xbb,0xc0,0xc9}.

This IPSec solution uses pre-shared keys in order to establish SAs. In other words, the key management scheme of the IPSec suite, namely the IKEv2 protocol has not been done. S.

Raza et al described in [16] their on-going research on Lightweight IKEv2.

```
/**
* \file
* AES-XCBC Message Authentication Code mode of operation
* \author
* Simon Duquennoy <simonduq@sics.se>
*/
#include <stdlib.h>
#include "net/uip.h"
#include "ipsec.h"
#include "aes-moo.h"
/*---------------------------------------------------------------------------*/
/* Pre-shared AES-XCBC-MAC keys, obtained from the main AES key as follows (from RFC 3566):
Derive 3 128-bit keys (K1, K2 and K3) from the 128-bit secret key K, as follows:
K1 = 0x01010101010101010101010101010101 encrypted with Key K
K2 = 0x02020202020202020202020202020202 encrypted with Key K
K3 = 0x03030303030303030303030303030303 encrypted with Key K

We provide a php script doing this in scripts/calculate_xcbc_keys.php
*/
static const unsigned char aesxcbc_k1[IPSEC_KEYSIZE] =
{0x3b,0xda,0x5b,0x6c,0x05,0x59,0x5d,0xe5,0x64,0x2b,0xf6,0x13,0xf8,0xd1,0xaf,0xd4};
static const unsigned char aesxcbc_k2[IPSEC_KEYSIZE] =
{0x74,0xf8,0x30,0xba,0x3a,0x1a,0x18,0x52,0x0d,0x2f,0xae,0x03,0x50,0xbe,0xb0,0x53};
static const unsigned char aesxcbc_k3[IPSEC_KEYSIZE] =
{0xca,0x6c,0x1f,0x23,0x64,0x70,0x37,0x07,0xb5,0xd1,0x8d,0x9b,0xc0,0x65,0x6b,0x43};
/*---------------------------------------------------------------------------*/
struct aes_xcbc_mac {
u8_t prev[IPSEC_KEYSIZE];
};
/*---------------------------------------------------------------------------*/
static void
aes_xcbc_mac_init(struct aes_xcbc_mac *axcbc, const unsigned char *key)
{
/* Set key */
CRYPTO_AES.init(key);
/* No previous block, set to 0 */
memset(axcbc->prev, 0, IPSEC_KEYSIZE);
}
/*---------------------------------------------------------------------------*/
static void
aes_xcbc_mac_step(struct aes_xcbc_mac *axcbc, unsigned char *buff)
{
int i;
/* prev ^= buff */
for(i=0; i<IPSEC_KEYSIZE; i++) {
```

```
axcbc->prev[i] ^= buff[i];
}
/* AES encrypt prev */
CRYPTO_AES.encrypt(axcbc->prev);
}
/*-----------------------------------------------------------------------------*/
static void
aes_xcbc_mac_final_step(struct aes_xcbc_mac *axcbc, unsigned char *buff, int len,
const unsigned char *key2, const unsigned char *key3)
{
int i;
unsigned char tmp[IPSEC_KEYSIZE];
/* the key is not the same if the last block isn't full */
unsigned char *key = (len == IPSEC_KEYSIZE) ? key2 : key3;
/* tmp = buff */
memcpy(tmp, buff, IPSEC_KEYSIZE);
/* add padding if needed */
for(i=0; i<IPSEC_KEYSIZE-len; i++) {
tmp[len+i] = (i == 0) ? 0x80 : 0x00;
}
/* lastinput ^= key */
for(i=0; i<IPSEC_KEYSIZE; i++) {
tmp[i] ^= key[i];
}
/* run normal step on tmp */
aes_xcbc_mac_step(axcbc, tmp);
}
/*-----------------------------------------------------------------------------*/
static void
aes_xcbc_mac_process(unsigned char *out, unsigned char *buff, int bufflen,
const unsigned char *key, const unsigned char *key2, const unsigned char *key3)
{
int i;
int len;
struct aes_xcbc_mac axcbc;
aes_xcbc_mac_init(&axcbc, key);
for(i=0; i< (bufflen-1)/IPSEC_KEYSIZE; i++) {
aes_xcbc_mac_step(&axcbc, buff + i * IPSEC_KEYSIZE);
}
len = bufflen % IPSEC_KEYSIZE;
aes_xcbc_mac_final_step(&axcbc, buff + i * IPSEC_KEYSIZE,
len == 0 ? IPSEC_KEYSIZE : len, key2, key3);
memcpy(out, axcbc.prev, IPSEC_MACSIZE);
}
/*-----------------------------------------------------------------------------*/
void
auth(unsigned char *out, unsigned char *buff, int bufflen)
```

```
{
aes_xcbc_mac_process(out, buff, bufflen, aesxcbc_k1, aesxcbc_k2, aesxcbc_k3);
}
/*---------------------------------------------------------------------------*/
struct ipsec_mac_implem aesxcbc_mac = {
auth,
};
/*---------------------------------------------------------------------------*/
```

Finally here is the usage with the Tmote sky from Linux. The border router M1 is compiled with the following commands:

```
cd contiki-2.6/examples/ipv6/rpl-border-router/
sudo make TARGET=sky border-router.upload MOTE=1
```

```
user@instant-contiki:~$ cd contiki-2.6/examples/ipv6/rpl-border-router/
user@instant-contiki:~/contiki-2.6/examples/ipv6/rpl-border-router$ sudo make TARGET=sky border-router.upload MOTE=1
[sudo] password for user:
msp430-objcopy border-router.sky -O ihex border-router.ihex
cp border-router.ihex tmpimage.ihex
make sky-u._dev_ttyUSB0
make[1]: Entering directory `/home/user/contiki-2.6/examples/ipv6/rpl-border-router'
+++++ Erasing /dev/ttyUSB0
MSP430 Bootstrap Loader Version: 1.39-telos-7
Use -h for help
Mass Erase...
Transmit default password ...
+++++ Programming /dev/ttyUSB0
MSP430 Bootstrap Loader Version: 1.39-telos-7
Invoking BSL...
Transmit default password ...
Current bootstrap loader version: 1.61 (Device ID: f16c)
Changing baudrate to 38400 ...
Program ...
46900 bytes programmed.
+++++ Resetting /dev/ttyUSB0
MSP430 Bootstrap Loader Version: 1.39-telos-7
Use -h for help
Reset device ...
make[1]: Leaving directory `/home/user/contiki-2.6/examples/ipv6/rpl-border-router'
rm border-router.ihex
```

The border router is initialized with the command:

```
make connect-router
```

```
user@instant-contiki:~/contiki-2.6/examples/ipv6/rpl-border-router$ make connect-router
using saved target 'sky'
```

```
sudo ../../../tools/tunslip6 aaaa::1/64

********SLIP started on ``/dev/ttyUSB0"

opened tun device ``/dev/tun0"

ifconfig tun0 inet `hostname` up

ifconfig tun0 add aaaa::1/64

ifconfig tun0 add fe80::0:0:0:1/64

ifconfig tun0


tun0 Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00

inet addr:127.0.1.1 P-t-P:127.0.1.1 Mask:255.255.255.255

inet6 addr: fe80::1/64 Scope:Link

inet6 addr: aaaa::1/64 Scope:Global

UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1

RX packets:0 errors:0 dropped:0 overruns:0 frame:0

TX packets:0 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:500

RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)


*** Address:aaaa::1 => aaaa:0000:0000:0000

Got configuration message of type P

Setting prefix aaaa::

Server IPv6 addresses:

aaaa::212:7400:1021:b4d8

fe80::212:7400:1021:b4d8
```

The application is now installed on the second mote M2 with the following command:

```
cd contiki-2.6/examples/ipv6/ipsec/

sudo make ipsec-example.upload MOTE=2
```

```
user@instant-contiki:~$ cd contiki-2.6/examples/ipv6/ipsec/

user@instant-contiki:~/contiki-2.6/examples/ipv6/ipsec$ sudo make ipsec-example.upload MOTE=2

msp430-gcc -DPROJECT_CONF_H=\"project-conf.h\" -DLOCAL_PLATFORM_CONF_H=1 -DCONTIKI=1 -
DCONTIKI_TARGET_SKY=1 -DUIP_CONF_IPV6=1 -DWITH_UIP6=1 -Os -fno-strict-aliasing -ffunction-sections -Wall -
mmcu=msp430f1611 -g -Idev -Ilib -Inet -Inet/mac -Inet/rime -Inet/rpl -Isys -Icfs -Ictk -Ilib/ctk -Iloader -I. -I../../../platform/sky/. -
I../../../platform/sky/dev -I../../../platform/sky/apps -I../../../platform/sky/net -I../../../cpu/msp430/f1xxx -I../../../cpu/msp430/. -
I../../../cpu/msp430/dev -I../../../core/dev -I../../../core/lib -I../../../core/net -I../../../core/net/mac -I../../../core/net/rime -I../../../core/net/rpl -
I../../../core/sys -I../../../core/cfs -I../../../core/ctk -I../../../core/lib/ctk -I../../../core/loader -I../../../core/. -Iipsec -MMD -c ipsec/aes-ctr.c -o
obj_sky/aes-ctr.o

ipsec/aes-ctr.c: In function ‘aes_ctr_init’:

ipsec/aes-ctr.c:36:3: warning: implicit declaration of function ‘memcpy’

ipsec/aes-ctr.c:36:3: warning: incompatible implicit declaration of built-in function ‘memcpy’

ipsec/aes-ctr.c: In function ‘aes_ctr_step’:

ipsec/aes-ctr.c:51:3: warning: incompatible implicit declaration of built-in function ‘memcpy’

ipsec/aes-ctr.c:53:3: warning: pointer targets in passing argument 1 of ‘cc2420_aes.encrypt’ differ in signedness

ipsec/aes-ctr.c:53:3: note: expected ‘unsigned char *’ but argument is of type ‘char *’

ipsec/aes-ctr.c: In function ‘aes_ctr_process’:

ipsec/aes-ctr.c:71:5: warning: pointer targets in passing argument 2 of ‘aes_ctr_step’ differ in signedness

ipsec/aes-ctr.c:42:1: note: expected ‘char *’ but argument is of type ‘unsigned char *’
```

```
ipsec/aes-ctr.c:73:3: warning: pointer targets in passing argument 2 of ‘aes_ctr_step’ differ in signedness

ipsec/aes-ctr.c:42:1: note: expected ‘char *’ but argument is of type ‘unsigned char *’

msp430-ar rcf contiki-sky.a obj_sky/rimeaddr.o obj_sky/timesynch.o obj_sky/rimestats.o obj_sky/cxmac.o obj_sky/xmac.o
obj_sky/nullmac.o obj_sky/lpp.o obj_sky/frame802154.o obj_sky/sicslowmac.o obj_sky/nullrdc.o obj_sky/nullrdc-noframer.o obj_sky/mac.o
obj_sky/framer-nullmac.o obj_sky/framer-802154.o obj_sky/csma.o obj_sky/contikimac.o obj_sky/phase.o obj_sky/rpl.o obj_sky/rpl-dag.o
obj_sky/rpl-icmp6.o obj_sky/rpl-timers.o obj_sky/rpl-of-etx.o obj_sky/rpl-ext-header.o obj_sky/process.o obj_sky/procinit.o
obj_sky/autostart.o obj_sky/elfloader.o obj_sky/profile.o obj_sky/timetable.o obj_sky/timetable-aggregate.o obj_sky/compower.o
obj_sky/serial-line.o obj_sky/memb.o obj_sky/mmem.o obj_sky/timer.o obj_sky/list.o obj_sky/etimer.o obj_sky/ctimer.o obj_sky/energest.o
obj_sky/rtimer.o obj_sky/stimer.o obj_sky/print-stats.o obj_sky/ifft.o obj_sky/crc16.o obj_sky/random.o obj_sky/checkpoint.o
obj_sky/ringbuf.o obj_sky/netstack.o obj_sky/uip-debug.o obj_sky/packetbuf.o obj_sky/queuebuf.o obj_sky/packetqueue.o obj_sky/uip6.o
obj_sky/tcpip.o obj_sky/psock.o obj_sky/uip-udp-packet.o obj_sky/uip-split.o obj_sky/resolv.o obj_sky/tcpdump.o obj_sky/uiplib.o
obj_sky/simple-udp.o obj_sky/uip-icmp6.o obj_sky/uip-nd6.o obj_sky/uip-packetqueue.o obj_sky/sicslowpan.o obj_sky/neighbor-attr.o
obj_sky/neighbor-info.o obj_sky/uip-ds6.o obj_sky/mt.o obj_sky/nullradio.o obj_sky/ipsec.o obj_sky/aes-ctr.o obj_sky/aes-xcbc-mac.o
obj_sky/cc2420-aes-mapping.o obj_sky/contiki-sky-platform.o obj_sky/sht11.o obj_sky/sht11-sensor.o obj_sky/light-sensor.o
obj_sky/battery-sensor.o obj_sky/button-sensor.o obj_sky/radio-sensor.o obj_sky/spi.o obj_sky/ds2411.o obj_sky/xmem.o obj_sky/i2c.o
obj_sky/node-id.o obj_sky/sensors.o obj_sky/cfs-coffee.o obj_sky/cc2420.o obj_sky/cc2420-aes.o obj_sky/cc2420-arch.o obj_sky/cc2420-
arch-sfd.o obj_sky/sky-sensors.o obj_sky/uip-ipchksum.o obj_sky/checkpoint-arch.o obj_sky/uart1.o obj_sky/slip_uart1.o obj_sky/uart1-
putchar.o obj_sky/me.o obj_sky/me_tabs.o obj_sky/slip.o obj_sky/msp430.o obj_sky/flash.o obj_sky/clock.o obj_sky/leds.o obj_sky/leds-
arch.o obj_sky/watchdog.o obj_sky/lpm.o obj_sky/mtarch.o obj_sky/rtimer-arch.o obj_sky/elfloader-msp430.o obj_sky/symtab.o
obj_sky/symbols.o

msp430-gcc -DPROJECT_CONF_H=\"project-conf.h\" -DLOCAL_PLATFORM_CONF_H=1 -DCONTIKI=1 -
DCONTIKI_TARGET_SKY=1 -DUIP_CONF_IPV6=1 -DWITH_UIP6=1 -Os -fno-strict-aliasing -ffunction-sections -Wall -
mmcu=msp430f1611 -g -Idev -Ilib -Inet -Inet/mac -Inet/rime -Inet/rpl -Isys -Icfs -Ictk -Ilib/ctk -Iloader -I. -I../../../platform/sky/. -
I../../../platform/sky/dev -I../../../platform/sky/apps -I../../../platform/sky/net -I../../../cpu/msp430/f1xxx -I../../../cpu/msp430/. -
I../../../cpu/msp430/dev -I../../../core/dev -I../../../core/lib -I../../../core/net -I../../../core/net/mac -I../../../core/net/rime -I../../../core/net/rpl -
I../../../core/sys -I../../../core/cfs -I../../../core/ctk -I../../../core/lib/ctk -I../../../core/loader -I../../../core/. -Iipsec -DAUTOSTART_ENABLE -c
ipsec-example.c -o ipsec-example.co

ipsec-example.c: In function ‘tcpip_handler’:

ipsec-example.c:86:5: warning: implicit declaration of function ‘uip_udp_packet_send’

ipsec-example.c:88:5: warning: implicit declaration of function ‘memset’

ipsec-example.c:88:5: warning: incompatible implicit declaration of built-in function ‘memset’

msp430-gcc -DPROJECT_CONF_H=\"project-conf.h\" -DLOCAL_PLATFORM_CONF_H=1 -DCONTIKI=1 -
DCONTIKI_TARGET_SKY=1 -DUIP_CONF_IPV6=1 -DWITH_UIP6=1 -Os -fno-strict-aliasing -ffunction-sections -Wall -
mmcu=msp430f1611 -g -Idev -Ilib -Inet -Inet/mac -Inet/rime -Inet/rpl -Isys -Icfs -Ictk -Ilib/ctk -Iloader -I. -I../../../platform/sky/. -
I../../../platform/sky/dev -I../../../platform/sky/apps -I../../../platform/sky/net -I../../../cpu/msp430/f1xxx -I../../../cpu/msp430/. -
I../../../cpu/msp430/dev -I../../../core/dev -I../../../core/lib -I../../../core/net -I../../../core/net/mac -I../../../core/net/rime -I../../../core/net/rpl -
I../../../core/sys -I../../../core/cfs -I../../../core/ctk -I../../../core/lib/ctk -I../../../core/loader -I../../../core/. -Iipsec -MMD -c
../../../platform/sky/./contiki-sky-main.c -o obj_sky/contiki-sky-main.o

msp430-gcc -mmcu=msp430f1611 -Wl,-Map=contiki-sky.map -Wl,--gc-sections,--undefined=_reset_vector__,--
undefined=InterruptVectors,--undefined=_copy_data_init__,--undefined=_clear_bss_init__,--undefined=_end_of_init__ ipsec-example.co
obj_sky/slip-bridge.o obj_sky/contiki-sky-main.o contiki-sky.a -o ipsec-example.sky

msp430-objcopy ipsec-example.sky -O ihex ipsec-example.ihex

cp ipsec-example.ihex tmpimage.ihex

make sky-u._dev_ttyUSB1

make[1]: Entering directory `/home/user/contiki-2.6/examples/ipv6/ipsec'

+++++ Erasing /dev/ttyUSB1

MSP430 Bootstrap Loader Version: 1.39-telos-7

Use -h for help

Mass Erase...

Transmit default password ...

+++++ Programming /dev/ttyUSB1

MSP430 Bootstrap Loader Version: 1.39-telos-7

Invoking BSL...

Transmit default password ...

Current bootstrap loader version: 1.61 (Device ID: f16c)

Changing baudrate to 38400 ...

Program ...

42798 bytes programmed.

+++++ Resetting /dev/ttyUSB1

MSP430 Bootstrap Loader Version: 1.39-telos-7
```

```
Use -h for help
Reset device ...
make[1]: Leaving directory `/home/user/contiki-2.6/examples/ipv6/ipsec'
rm ipsec-example.co obj_sky/contiki-sky-main.o ipsec-example.ihex
```

The user should finally execute the commands to test the application:

```
cd ~/contiki-2.6/examples/ipv6/ipsec/scripts
./client.py aaaa::212:7400:10f4:103
```

We made some experiments with AH, ESP, and AH+ESP.

Data are composed by a succession of letter 32 ‘a’

Case A: AH only

ipsec-tools.conf

```
#!/usr/sbin/setkey -f

# NOTE: Do not use this file if you use racoon with racoon-tool
# utility. racoon-tool will setup SAs and SPDs automatically using
# /etc/racoon/racoon-tool.conf configuration.
#

# Configuration for aaaa::1

# Flush the SAD and SPD
flush;
spdflush;

# Attention: Use this keys only for testing purposes!
# Generate your own keys!

# AH with AES-XCBC-MAC using 128-bit key
add aaaa::1 aaaa::0212:7400:13b7:7659 ah 0x0001 -A aes-xcbc-mac 0xcf5faaca70ee5ec4c8f43158a45c0363;
add aaaa::0212:7400:13b7:7659 aaaa::1 ah 0x0001 -A aes-xcbc-mac 0xcf5faaca70ee5ec4c8f43158a45c0363;

add aaaa::1 aaaa::0212:7400:13b7:7f30 ah 0x0002 -A aes-xcbc-mac 0xcf5faaca70ee5ec4c8f43158a45c0363;
add aaaa::0212:7400:13b7:7f30 aaaa::1 ah 0x0002 -A aes-xcbc-mac 0xcf5faaca70ee5ec4c8f43158a45c0363;

# Security policies
spdadd aaaa::1 aaaa::0/64 any -P out ipsec
ah/transport//require;

spdadd aaaa::0/64 aaaa::1 any -P in ipsec
ah/transport//require;
```

```
No.     Time           Source                Destination           Protocol Length Info
1 0.000000000    aaaa::1               aaaa::212:7400:13b7:7f30 UDP      104    Source port: 36868  Destination port: search-agent

Frame 1: 104 bytes on wire (832 bits), 104 bytes captured (832 bits) on interface 0
Raw packet data
   No link information available
Internet Protocol Version 6, Src: aaaa::1 (aaaa::1), Dst: aaaa::212:7400:13b7:7f30(aaaa::212:7400:13b7:7f30)
0110 .... = Version: 6
   .... 0000 0000 ........ .... .... .... = Traffic class: 0x00000000
.... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
   Payload length: 64
   Next header: AH (51)
   Hop limit: 64
   Source: aaaa::1 (aaaa::1)
Destination: aaaa::212:7400:13b7:7f30 (aaaa::212:7400:13b7:7f30)
   Authentication Header
User Datagram Protocol, Src Port: 36868 (36868), Dst Port: search-agent (1234)
Source port: 36868 (36868)
   Destination port: search-agent (1234)
Length: 40
   Checksum: 0xf661 [validation disabled]
Data (32 bytes)

0000  61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61   aaaaaaaaaaaaaaaa
0010  61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61   aaaaaaaaaaaaaaaa
   Data: 61616161616161616161616161616161616161616161616161...

[Length: 32]
```

```
0000  60 00 00 00 00 40 33 40  aa aa 00 00 00 00 00 00   `....@3@ ........
0010  00 00 00 00 00 00 00 01  aa aa 00 00 00 00 00 00   ........ ........
0020  02 12 74 00 13 b7 7f 30  11 04 00 00 00 00 00 02   ..t....0 ........
0030  00 00 00 01 e0 f5 e6 f7  33 89 1a 1a 36 14 7e d6   ........ 3...6.~.
0040  90 04 04 d2 00 28 f6 61  61 61 61 61 61 61 61 61   .....(.a aaaaaaaa
0050  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
0060  61 61 61 61 61 61 61 61                           aaaaaaaa
```

```
No.     Time           Source                Destination           Protocol Length Info
     2 0.200879000    aaaa::212:7400:13b7:7f30 aaaa::1               UDP      103    Source port: 0  Destination port: 289 [BAD UDP LENGTH
31558 > IP PAYLOAD LENGTH]

Frame 2: 103 bytes on wire (824 bits), 103 bytes captured (824 bits) on interface 0
Raw packet data
   No link information available
Internet Protocol Version 6, Src: aaaa::212:7400:13b7:7f30 (aaaa::212:7400:13b7:7f30), Dst: aaaa::1 (aaaa::1)
0110 .... = Version: 6
   .... 0000 0000 ........ .... .... .... = Traffic class: 0x00000000
.... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
   Payload length: 63
   Next header: AH (51)
   Hop limit: 63
   Source: aaaa::212:7400:13b7:7f30 (aaaa::212:7400:13b7:7f30)
Destination: aaaa::1 (aaaa::1)
   Authentication Header
User Datagram Protocol, Src Port: 0 (0), Dst Port: 289 (289)
Source port: 0 (0)
Destination port: 289 (289)
   Length: 31558 (bogus, payload length 55)
      [Expert Info (Error/Malformed): Bad length value 31558 > IP payload length]
   Checksum: 0x5b60 [unchecked, not all data available]
Data (47 bytes)

0000  2c 6c 82 f5 24 11 1e 04 d2 90 04 00 28 e6 51 62   ,l..$.......(.Qb
0010  62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62   bbbbbbbbbbbbbbbb
0020  62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62   bbbbbbbbbbbbbbbb
   Data: 2c6c82f524111e04d290040028e651626262626262626262...
   [Length: 47]
```

```
0000  60 00 00 00 00 3f 33 3f  aa aa 00 00 00 00 00 00   `....?3? ........
0010  02 12 74 00 13 b7 7f 30  aa aa 00 00 00 00 00 00   ..t....0 ........
0020  00 00 00 00 00 00 00 01  11 00 00 00 00 00 02 00   ........ ........
0030  00 00 01 21 7b 46 5b 60  2c 6c 82 f5 24 11 1e 04   ...!{F[` ,l..$...
0040  d2 90 04 00 28 e6 51 62  62 62 62 62 62 62 62 62   ....(.Qb bbbbbbbb
```

```
0050  62 62 62 62 62 62 62 62  62 62 62 62 62 62 62 62   bbbbbbbb bbbbbbbb
0060  62 62 62 62 62 62 62                               bbbbbbb
```

Case B: ESP only

ipsec-tools.conf

```
#!/usr/sbin/setkey -f

# NOTE: Do not use this file if you use racoon with racoon-tool
# utility. racoon-tool will setup SAs and SPDs automatically using
# /etc/racoon/racoon-tool.conf configuration.
#

# Configuration for aaaa::1

# Flush the SAD and SPD
flush;
spdflush;

# Attention: Use this keys only for testing purposes!
# Generate your own keys!

# ESP AES-CTR using 160-bit key (128 for aes key, 32 for ctr nonce)
add aaaa::1 aaaa::0212:7400:13b7:7659 esp 0x0001 -E aes-ctr 0xcf5faaca70ee5ec4c8f43158a45c036369bbc0c9 -A aes-xcbc-mac
0xcf5faaca70ee5ec4c8f43158a45c0363;
add aaaa::0212:7400:13b7:7659 aaaa::1 esp 0x0001 -E aes-ctr 0xcf5faaca70ee5ec4c8f43158a45c036369bbc0c9 -A aes-xcbc-mac
0xcf5faaca70ee5ec4c8f43158a45c0363;

add aaaa::1 aaaa::0212:7400:13b7:7f30 esp 0x0002 -E aes-ctr 0xcf5faaca70ee5ec4c8f43158a45c036369bbc0c9 -A aes-xcbc-mac
0xcf5faaca70ee5ec4c8f43158a45c0363;
add aaaa::0212:7400:13b7:7f30 aaaa::1 esp 0x0002 -E aes-ctr 0xcf5faaca70ee5ec4c8f43158a45c036369bbc0c9 -A aes-xcbc-mac
0xcf5faaca70ee5ec4c8f43158a45c0363;

# Security policies
spdadd aaaa::1 aaaa::0/64 any -P out ipsec
esp/transport//require;

spdadd aaaa::0/64 aaaa::1 any -P in ipsec
esp/transport//require;
```

```
No.     Time           Source                Destination           Protocol Length Info
      1 0.000000000    aaaa::1               aaaa::212:7400:13b7:7f30 ESP     112    ESP (SPI=0x00000002)

Frame 1: 112 bytes on wire (896 bits), 112 bytes captured (896 bits) on interface 0
    Interface id: 0
    WTAP_ENCAP: 7
    Arrival Time: Aug  1, 2013 15:24:37.264340000 CEST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1375363477.264340000 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 112 bytes (896 bits)
```

```
    Capture Length: 112 bytes (896 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: raw:ipv6:esp]
Raw packet data
    No link information available
Internet Protocol Version 6, Src: aaaa::1 (aaaa::1), Dst: aaaa::212:7400:13b7:7f30 (aaaa::212:7400:13b7:7f30)
0110 .... = Version: 6
    .... 0000 0000 ........ .... .... .... = Traffic class: 0x00000000
.... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 72
    Next header: ESP (50)
    Hop limit: 64
    Source: aaaa::1 (aaaa::1)
Destination: aaaa::212:7400:13b7:7f30 (aaaa::212:7400:13b7:7f30)
Encapsulating Security Payload
ESP SPI: 0x00000002 (2)
    ESP Sequence: 1
```

```
0000  60 00 00 00 00 48 32 40  aa aa 00 00 00 00 00 00   `....H2@ ........
0010  00 00 00 00 00 00 00 01  aa aa 00 00 00 00 00 00   ........ ........
0020  02 12 74 00 13 b7 7f 30  00 00 00 02 00 00 00 01   ..t....0 ........
0030  0b 10 89 7f 7a 6d da 89  10 6a 90 17 24 64 d5 00   ....zm.. .j..$d..
0040  7b 48 ea 75 85 62 78 00  14 f9 37 a7 1d 90 58 7e   {H.u.bx. ..7...X~
0050  61 76 3e 26 af 7f 0e 84  22 05 be 1d 96 46 5e 32   av>&.... "....F^2

0060  5f 63 47 cb 1d b7 6c 49  05 db ed c8 69 b8 b5 36   _cG...lI ....i..6
```

```
No.     Time           Source                Destination           Protocol Length Info
      2 0.165737000    aaaa::212:7400:13b7:7f30 aaaa::1               ESP      112    ESP (SPI=0x00000002)

Frame 2: 112 bytes on wire (896 bits), 112 bytes captured (896 bits) on interface 0
    Interface id: 0
    WTAP_ENCAP: 7
    Arrival Time: Aug  1, 2013 15:24:37.430077000 CEST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1375363477.430077000 seconds
    [Time delta from previous captured frame: 0.165737000 seconds]
    [Time delta from previous displayed frame: 0.165737000 seconds]
    [Time since reference or first frame: 0.165737000 seconds]
    Frame Number: 2
    Frame Length: 112 bytes (896 bits)
    Capture Length: 112 bytes (896 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: raw:ipv6:esp]
Raw packet data
    No link information available
Internet Protocol Version 6, Src: aaaa::212:7400:13b7:7f30 (aaaa::212:7400:13b7:7f30), Dst: aaaa::1 (aaaa::1)
0110 .... = Version: 6
    .... 0000 0000 ........ .... .... .... = Traffic class: 0x00000000
.... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 72
    Next header: ESP (50)
    Hop limit: 63
    Source: aaaa::212:7400:13b7:7f30 (aaaa::212:7400:13b7:7f30)
    Destination: aaaa::1 (aaaa::1)
Encapsulating Security Payload
ESP SPI: 0x00000002 (2)
    ESP Sequence: 1
```

```
0000  60 00 00 00 00 48 32 3f  aa aa 00 00 00 00 00 00   `....H2? ........
0010  02 12 74 00 13 b7 7f 30  aa aa 00 00 00 00 00 00   ..t....0 ........
0020  00 00 00 00 00 00 00 01  00 00 00 02 00 00 00 01   ........ ........
0030  01 00 00 00 00 00 00 00  bb af c8 4b 92 01 9b d7   ........ ...K....
0040  9c 83 65 b0 bc d8 3f 23  3d fb 21 e8 d6 df 8b a4   ..e...?# =.!.....
0050  56 7f b2 cc f4 38 50 6b  0b f3 43 31 ca 23 26 d1   V....8Pk ..C1.#&.

0060  9f 57 ec d6 fe b2 9a 16  c5 96 fc 31 15 92 ff f5   .W...... ...1....
```

./client.py aaaa::0212:7400:13b7:7f30

Duration: 165 ms

Request: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Response: bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

Case C: AH + ESP

ipsec-tools.conf

```
#!/usr/sbin/setkey -f

# NOTE: Do not use this file if you use racoon with racoon-tool
# utility. racoon-tool will setup SAs and SPDs automatically using
# /etc/racoon/racoon-tool.conf configuration.
#

# Configuration for aaaa::1

# Flush the SAD and SPD
flush;
spdflush;

# Attention: Use this keys only for testing purposes!
# Generate your own keys!

# ESP AES-CTR using 160-bit key (128 for aes key, 32 for ctr nonce)
add aaaa::1 aaaa::0212:7400:13b7:7659 esp 0x0001 -E aes-ctr 0xcf5faaca70ee5ec4c8f43158a45c036369bbc0c9 -A aes-xcbc-mac
0xcf5faaca70ee5ec4c8f43158a45c0363;
add aaaa::0212:7400:13b7:7659 aaaa::1 esp 0x0001 -E aes-ctr 0xcf5faaca70ee5ec4c8f43158a45c036369bbc0c9 -A aes-xcbc-mac
0xcf5faaca70ee5ec4c8f43158a45c0363;

add aaaa::1 aaaa::0212:7400:13b7:7f30 esp 0x0002 -E aes-ctr 0xcf5faaca70ee5ec4c8f43158a45c036369bbc0c9 -A aes-xcbc-mac
0xcf5faaca70ee5ec4c8f43158a45c0363;
add aaaa::0212:7400:13b7:7f30 aaaa::1 esp 0x0002 -E aes-ctr 0xcf5faaca70ee5ec4c8f43158a45c036369bbc0c9 -A aes-xcbc-mac
0xcf5faaca70ee5ec4c8f43158a45c0363;

# Security policies
spdadd aaaa::1 aaaa::0/64 any -P out ipsec
esp/transport//require;

spdadd aaaa::0/64 aaaa::1 any -P in ipsec
esp/transport//require;
```

```
No.     Time           Source                Destination           Protocol Length Info
     1 0.000000000    aaaa::1               aaaa::212:7400:13b7:7f30 ESP      136    ESP (SPI=0x00000004)

Frame 1: 136 bytes on wire (1088 bits), 136 bytes captured (1088 bits) on interface 0
    Interface id: 0
    WTAP_ENCAP: 7
    Arrival Time: Aug  6, 2013 10:34:10.301278000 CEST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1375778050.301278000 seconds
```

```
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 136 bytes (1088 bits)
    Capture Length: 136 bytes (1088 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: raw:ipv6:esp]
Raw packet data
    No link information available
Internet Protocol Version 6, Src: aaaa::1 (aaaa::1), Dst: aaaa::212:7400:13b7:7f30 (aaaa::212:7400:13b7:7f30)
0110 .... = Version: 6
    .... 0000 0000 ........ .... .... .... = Traffic class: 0x00000000
.... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 96
    Next header: AH (51)
    Hop limit: 64
    Source: aaaa::1 (aaaa::1)
Destination: aaaa::212:7400:13b7:7f30 (aaaa::212:7400:13b7:7f30)
Authentication Header
Encapsulating Security Payload
ESP SPI: 0x00000004 (4)
    ESP Sequence: 5
```

```
0000  60 00 00 00 00 60 33 40  aa aa 00 00 00 00 00 00   `....`3@ ........
0010  00 00 00 00 00 00 00 01  aa aa 00 00 00 00 00 00   ........ ........
0020  02 12 74 00 13 b7 7f 30  32 04 00 00 00 00 00 02   ..t....0 2.......
0030  00 00 00 05 b5 30 08 45  70 7d be f4 22 7d 46 e5   .....0.E p}.."}F.
0040  00 00 00 04 00 00 00 05  35 45 e0 ba eb 4b a0 55   ........ 5E...K.U
0050  b2 ba fd 57 7a de dc 17  d5 01 13 6c ce 39 88 36   ...Wz... ...l.9.6
0060  12 1c 3c e2 74 0c ad 00  28 eb b3 89 13 20 65 24   ..<.t... (.... e$
0070  a4 1a 1c 54 ad c5 6e 72  2c 69 99 1a 3d 95 f5 d5   ...T..nr ,i..=...

0080  22 0f 86 8e 01 ad 0b 92                           ".......
```

```
No.     Time           Source                Destination           Protocol Length Info
      2 0.240065000    aaaa::212:7400:13b7:7f30 aaaa::1               ESP      112    ESP (SPI=0x00000004)

Frame 2: 112 bytes on wire (896 bits), 112 bytes captured (896 bits) on interface 0
    Interface id: 0
    WTAP_ENCAP: 7
    Arrival Time: Aug  6, 2013 10:34:10.541343000 CEST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1375778050.541343000 seconds
    [Time delta from previous captured frame: 0.240065000 seconds]
    [Time delta from previous displayed frame: 0.240065000 seconds]
    [Time since reference or first frame: 0.240065000 seconds]
    Frame Number: 2
    Frame Length: 112 bytes (896 bits)
    Capture Length: 112 bytes (896 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: raw:ipv6:esp]
Raw packet data
    No link information available
Internet Protocol Version 6, Src: aaaa::212:7400:13b7:7f30 (aaaa::212:7400:13b7:7f30), Dst: aaaa::1 (aaaa::1)
0110 .... = Version: 6
    .... 0000 0000 ........ .... .... .... = Traffic class: 0x00000000
.... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 72
    Next header: ESP (50)
    Hop limit: 63
    Source: aaaa::212:7400:13b7:7f30 (aaaa::212:7400:13b7:7f30)
    Destination: aaaa::1 (aaaa::1)
Encapsulating Security Payload
ESP SPI: 0x00000004 (4)
    ESP Sequence: 3
```

```
0000  60 00 00 00 00 48 32 3f  aa aa 00 00 00 00 00 00   `....H2? ........
0010  02 12 74 00 13 b7 7f 30  aa aa 00 00 00 00 00 00   ..t....0 ........
0020  00 00 00 00 00 00 00 01  00 00 00 04 00 00 00 03   ........ ........
0030  03 00 00 00 00 00 00 00  d0 d5 34 70 e8 47 c2 2b   ........ ..4p.G.+
```

```
0040  8c 07 0b 5a 74 a5 65 34  1b 8f 3c 87 1d 54 58 e3   ...Zt.e4 ..<..TX.
0050  6d 41 79 1b 37 67 b7 4b  10 12 f7 2c a0 0f 1c 00   mAy.7g.K ...,....

0060  e5 4a 8a fb 66 99 3d 4c  5b 7b 52 ad 7c a1 a4 09   .J..f.=L [{R.|...
```

The compressed version of IPSec has been implemented for the Contiki OS. For the AH Authentication, the mode HMAC-SHA1-96 needs 24 Bytes (possibly reduced to 16 Bytes for compressed IPSec). In comparison, 802.15.4 in AES-CBC-MAC-96 mode needs 12 Bytes. For the ESP Encryption, AES-CBC needs 18 Bytes (possibly reduced to 12 Bytes for compressed IPSec). In comparison, 802.15.4 in AES-CTR mode needs 5 Bytes. For the ESP Encryption and Authentication, the mode AES-CBC and HMAC-SHA1-96 needs 30 Bytes (possibly reduced to 24 Bytes). In comparison, 802.15.4 in AES-CCM-128 needs 21 Bytes.

The authors provided in their publication [15] the ROM and RAM footprints of their IPSec solution (see Table 22).

System	ROM overall (kB)	ROM overhead (kB)	RAM overall (kB)	RAM additional (kB)
Without IPSec	32.9	-	8.0	-
AH HMAC-SHA1-96	36.8	3.9	9.1	1.1
AH XCBC-MAC-96	38.4	5.5	8.5	0.5
ESP AES-CBC	41.4	8.5	8.3	0.3
ESP AES-CTR	39.8	6.9	9.1	0.3
ESP AES-XCBC-MAC-96	39.8	6.9	8.3	0.3
ESP AES-CBC + AES-XCBC-MAC-96	41.9	9	8.3	0.3
ESP AES-CTR + AES-XCBC-MAC-96	40.3	7.4	8.3	0.3

Table 22: ROM and RAM footprints (IPSec)

The overhead memory footprint of the IPSec implementation ranges from 3.9 kB to 9 kB ROM and 0.3 kB to 1.1 kB RAM depending on the used protocol and mode of operation.

For the energy overhead, authors showed that ESP consumes more energy than AH. AES-CBC and AES-CTR provides similar performance and energy consumption. The energy consumption grows linearly with the data size.

3.1.6.9 TLS/DTLS in Contiki

V. Perelman proposed in [17] to secure IPv6-enabled wireless sensor networks according to an implementation of TLS/DTLS for the Contiki OS, based on a pre-shared key cipher suite (TLS_PSK_WITH_AES_128_CCM_8). TLS relies on the connection-oriented Transport Control Protocol (TCP). It is based on Secure Sockets Layer (SSL).

The Record Protocol is used to fragment data into blocks called records. It is associated with the following protocols:

- The Handshake protocol negotiates the encryption and compression algorithms for communication. Nodes are authenticated and the keys needed for encryption are established.
- The ChangeCipherSpec protocol consists of a single message sent during the Handshake protocol. It informs that all next messages are encrypted and compressed using the negotiated algorithms and keys.
- The Alert protocol notifies any abnormal execution of TLS.
- The Application Data protocol finally transports the encrypted application data.

Datagram Transport Layer Security (DTLS) is an equivalent of TLS for datagram based

applications. Two extra fields are added into DTLS records' header, e.g. the epoch number used to specify which cipher state is being used to protect the record payload and the sequence number used to handle lost or delivered out of order records.

The source code of the library is publicly available under the BSD 2-Clause License and can be found at the CNDS website:

cnds.eecs.jacobs-university.de/software

The following steps have to be taken in order to integrate and use the TLS/DTLS libraries. The tls and dtls folders that contain the code to the core/net folder should be copied within the Contiki OS. The file **Makefile.include**, located at the root of the Contiki OS, is modified as what follows:

```
...
include $(CONTIKI)/core/net/rime/Makefile.rime
include $(CONTIKI)/core/net/mac/Makefile.mac
#TLS/DTLS change added below
ifdef TLS
include $(CONTIKI)/core/net/tls/Makefile.tls
endif
ifdef DTLS
include $(CONTIKI)/core/net/dtls/Makefile.dtls
endif
#TLS/DTLS change done
...
CONTIKI_SOURCEFILES += $(CONTIKIFILES)
CONTIKIDIRS += ${addprefix $(CONTIKI)/core/,dev lib net net/mac net/rime \
net/rpl sys cfs ctk lib/ctk loader . }
#TLS/DTLS change added below
ifdef TLS
CONTIKIDIRS += $(CONTIKI)/core/net/tls
endif
ifdef DTLS
CONTIKIDIRS += $(CONTIKI)/core/net/dtls
endif
#TLS/DTLS change done
...
```

The user can create an application that requires the use of the libraries within the examples folder of the Contiki OS. A simple client-server application **test-tls/test-dtls** can also be found in the CNDS website. Inside the Makefile of the application, the user must include the relevant library (<core/net/tls.h> or <core/net/dtls.h>) and also set TLS=1 or DTLS=1.

3.1.6.10 Tinydtls in Contiki

O. Bergmann designed a specific library enabling a datagram server with DTLS support [18]. The current release is tinydtls-0.4.0 (July 2013). It also provides an application example. The source files are available here: http://sourceforge.net/projects/tinydtls/files/.

We installed tinydtls on the Instant Contiki 2.6 virtual machine. The set-up is done as follows. The source file should be uncompressed in a defined location.

The configuration is done with the command:

```
./configure
```

Thereafter we perform the installation.

```
make

sudo make install.
```

The user just needs to add tinydtls to the variable APPS in the Makefile.

We finally run server and client applications in 2 consoles:

```
user@instant-contiki:~/Desktop/tinydtls-0.3.1/tests$ ./dtls-client -o results.txt -p 20220 localhost
Jul 03 13:01:07 ALRT setsockopt IPV6_PKTINFO: Protocol not available
dtls_new_peer: 127.0.0.1:20220
1000000002004efc7f0000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000 16 FE FD 00 00 00 00 00  00 00 00 00 36
00000000 01 00 00 2A 00 00 00 00  00 00 00 2A FE FD 51 D4
00000010 04 73 64 74 F9 F6 76 1E  51 4E 34 1A 9B E5 0A 60
00000020 BC DC D8 0D 5A A8 44 4E  D0 0F 2B 9B 6A 1E 00 00
00000030 00 02 00 FD 01 00
1000000002004efc7f0000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
16fefd0000000000000000001f030000130000000000000013fefd10692a31fe7dfa39fcfa1a18f78e1c64fc
dtls_handle_message: FOUND PEER
00000000 16 FE FD 00 00 00 00 00  00 00 00 00 1F
00000000 03 00 00 13 00 00 00 00  00 00 00 13 FE FD 10 69
00000010 2A 31 FE 7D FA 39 FC FA  1A 18 F7 8E 1C 64 FC
add MAC data:
0100003a000100000000003afefd51d404736474f9f6761e514e341a9be50a60bcdcd80d5aa8444ed00f2b9b6a1e0010692a31fe7dfa39fcfa1a18f
78e1c64fc000200fd0100
00000000 16 FE FD 00 00 00 00 00  00 00 01 00 46
00000000 01 00 00 3A 00 01 00 00  00 00 00 3A FE FD 51 D4
00000010 04 73 64 74 F9 F6 76 1E  51 4E 34 1A 9B E5 0A 60
00000020 BC DC D8 0D 5A A8 44 4E  D0 0F 2B 9B 6A 1E 00 10
00000030 69 2A 31 FE 7D FA 39 FC  FA 1A 18 F7 8E 1C 64 FC
00000040 00 02 00 FD 01 00
1000000002004efc7f0000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
16fefd00000000000000010032020000260001000000000026fefd7d6a95e707584ae72f69e5c726a8ccd49d1ebda097b94bfcdde1537f51d4047
30000fd0016fefd0000000000000002000c0e0000000002000000000000
dtls_handle_message: FOUND PEER
00000000 16 FE FD 00 00 00 00 00  00 00 01 00 32
00000000 02 00 00 26 00 01 00 00  00 00 00 26 FE FD 7D 6A
00000010 95 E7 07 58 4A E7 2F 69  E5 C7 26 A8 CC D4 9D 1E
00000020 BD A0 97 B9 4B FC DD E1  53 7F 51 D4 04 73 00 00
00000030 FD 00
add MAC data: 020000260001000000000026fefd7d6a95e707584ae72f69e5c726a8ccd49d1ebda097b94bfcdde1537f51d404730000fd00
client_random: 51 d4 04 73 64 74 f9 f6 76 1e 51 4e 34 1a 9b e5 0a 60 bc dc d8 0d 5a a8 44 4e d0 0f 2b 9b 6a 1e
server_random: 7d 6a 95 e7 07 58 4a e7 2f 69 e5 c7 26 a8 cc d4 9d 1e bd a0 97 b9 4b fc dd e1 53 7f 51 d4 04 73
psk: (9 bytes):00000000 73 65 63 72 65 74 50 53  4B
pre_master_secret: (22 bytes): 00 09 00 00 00 00 00 00 00 00 00 00 00 09 73 65 63 72 65 74 50 53 4b
master_secret (48 bytes): 7e 6c c0 64 59 be 2c 08 4d f1 e1 6d 19 ba cf aa 15 2d d1 ca 5a 78 9e 9e 76 2c a6 21 ec 0b 69 83 00 b7 da 8d 66 3f
13 b4 bc 9e f0 91 fc b8 95 9e
key_block (40 bytes):
client_MAC_secret:
server_MAC_secret:
client_write_key:       8b3257d1c74f8599ba53f2c9cbbc8768
server_write_key:       f5954dbf6d310962c0263eeb2ff53622
client_IV:              3006e48e
server_IV:              de85c680
00000000 16 FE FD 00 00 00 00 00  00 00 02 00 0C
00000000 0E 00 00 00 00 02 00 00  00 00 00 00
add MAC data: 0e00000000020000000000000
add MAC data: 100000110002000000000011000f436c69656e745f6964656e74697479
00000000 16 FE FD 00 00 00 00 00  00 00 02 00 1D
00000000 10 00 00 11 00 02 00 00  00 00 00 11 00 0F 43 6C
00000010 69 65 6E 74 5F 69 64 65  6E 74 69 74 79
00000000 14 FE FD 00 00 00 00 00  00 00 03 00 01
00000000 01
key_block:
```

client_MAC_secret:
server_MAC_secret:
client_write_key: 8b3257d1c74f8599ba53f2c9cbbc8768
server_write_key: f5954dbf6d310962c0263eeb2ff53622
client_IV: 3006e48e
server_IV: de85c680
add MAC data: 1400000c000300000000000c3d83b7d0c236609be85879ca
nonce: 3006e48e000100000000000000000000
key: 8b3257d1c74f8599ba53f2c9cbbc8768
00010000000000003bc4ef89a08980c8884f378eb5faa96cdd1606a99fb0e1b714a442c007fbee3d
00000000 16 FE FD 00 01 00 00 00 00 00 00 00 28
00000000 14 00 00 0C 00 03 00 00 00 00 00 0C 3D 83 B7 D0
00000010 C2 36 60 9B E8 58 79 CA
1000000002004efc7f000001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000
14fefd000000000000003000101
dtls_handle_message: FOUND PEER
1000000002004efc7f000001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000
16fefd000100000000000000280001000000000000c1d17e5e2820908cd2a74b43d216e43a7656d659b59291b21636b6290cdd6ba6
dtls_handle_message: FOUND PEER
nonce: de85c680000100000000000000000000
key: f5954dbf6d310962c0263eeb2ff53622
ciphertext:
c1d17e5e2820908cd2a74b43d216e43a7656d659b59291b21636b6290cdd6ba6
decrypt_verify(): found 24 bytes cleartext

cleartext:
1400000c000300000000000c1478e52425329b327b305603
00000000 16 FE FD 00 01 00 00 00 00 00 00 00 28
00000000 14 00 00 0C 00 03 00 00 00 00 00 0C 14 78 E5 24
00000010 25 32 9B 32 7B 30 56 03
d: 1478e52425329b327b305603
v: 1478e52425329b327b305603

user@instant-contiki:~/Desktop/tinydtls-0.3.1/tests$ ./dtls-server
dtls_handle_message: PEER NOT FOUND
[::ffff:127.0.0.1]:60150
1c0000000a00eaf60000000000000000000000000000ffff7f000001000000008bf276b7d02a78b748fb75b70100000000100000000000000094a7
5db7ec8374b75c35fbbfec17050800000000dc840408246005080100000 0d2217a5f1837fbbfc8cd5ab7f47f74b71837fbbf00000000d035fbbf50
36fbbfd036fbbf1837fbbf5036fbbfd035fbbf
692a31fe7dfa39fcfa1a18f78e1c64fc

Jul 03 13:01:07 WARN cannot verify peer
dtls_handle_message: PEER NOT FOUND
[::ffff:127.0.0.1]:60150
1c0000000a00eaf60000000000000000000000000000ffff7f000001000000008bf276b7d02a78b748fb75b70100000000100000000000000094a7
5db7ec8374b75c35fbbfec17050800000000dc840408246005080100000 0d2217a5f1837fbbfc8cd5ab7f47f74b71837fbbf00000000d035fbbf50
36fbbfd036fbbf1837fbbf5036fbbfd035fbbf
692a31fe7dfa39fcfa1a18f78e1c64fc
692a31fe7dfa39fcfa1a18f78e1c64fc
dtls_new_peer: [::ffff:127.0.0.1]:60150
1c0000000a00eaf60000000000000000000000000000ffff7f000001000000008bf276b7d02a78b748fb75b70100000000100000000000000094a7
5db7ec8374b75c35fbbfec17050800000000dc840408246005080100000 0d2217a5f1837fbbfc8cd5ab7f47f74b71837fbbf00000000d035fbbf50
36fbbfd036fbbf1837fbbf5036fbbfd035fbbf
add MAC data:
0100003a000100000000003afefd51d404736474f9f6761e514e341a9be50a60bcdcd80d5aa8444ed00f2b9b6a1e0010692a31fe7dfa39fcfa1a18f
78e1c64fc000200fd0100
client_random: 51 d4 04 73 64 74 f9 f6 76 1e 51 4e 34 1a 9b e5 0a 60 bc dc d8 0d 5a a8 44 4e d0 0f 2b 9b 6a 1e
server_random: 7d 6a 95 e7 07 58 4a e7 2f 69 e5 c7 26 a8 cc d4 9d 1e bd a0 97 b9 4b fc dd e1 53 7f 51 d4 04 73
psk: (9 bytes):00000000 73 65 63 72 65 74 50 53 4B
pre_master_secret: (22 bytes): 00 09 00 00 00 00 00 00 00 00 00 09 73 65 63 72 65 74 50 53 4b
master_secret (48 bytes): 7e 6c c0 64 59 be 2c 08 4d f1 e1 6d 19 ba cf aa 15 2d d1 ca 5a 78 9e 9e 76 2c a6 21 ec 0b 69 83 00 b7 da 8d 66 3f
13 b4 bc 9e f0 91 fc b8 95 9e
key_block (40 bytes):
client_MAC_secret:
server_MAC_secret:
client_write_key: 8b3257d1c74f8599ba53f2c9cbbc8768
server_write_key: f5954dbf6d310962c0263eeb2ff53622
client_IV: 3006e48e
server_IV: de85c680
add MAC data: 020000260001000000000026fefd7d6a95e707584ae72f69e5c726a8ccd49d1ebda097b94bfcdde1537f51d404730000fd00
add MAC data: 0e0000000002000000000000
dtls_handle_message: FOUND PEER
00000000 16 FE FD 00 00 00 00 00 00 00 02 00 1D
00000000 10 00 00 11 00 02 00 00 00 00 00 11 00 0F 43 6C

00000010 69 65 6E 74 5F 69 64 65 6E 74 69 74 79
add MAC data: 1000001100020000000000011000f436c69656e745f6964656e74697479
dtls_handle_message: FOUND PEER
00000000 14 FE FD 00 00 00 00 00 00 00 03 00 01
00000000 01
00000000 14 FE FD 00 00 00 00 00 00 00 03 00 01
00000000 01
key_block:
client_MAC_secret:
server_MAC_secret:
client_write_key: 8b3257d1c74f8599ba53f2c9cbbc8768
server_write_key: f5954dbf6d310962c0263eeb2ff53622
client_IV: 3006e48e
server_IV: de85c680
dtls_handle_message: FOUND PEER
nonce: 3006e48e000100000000000000000000
key: 8b3257d1c74f8599ba53f2c9cbbc8768
ciphertext:
3bc4ef89a08980c8884f378eb5faa96cdd1606a99fb0e1b714a442c007fbee3d
decrypt_verify(): found 24 bytes cleartext

cleartext:
1400000c000300000000000c3d83b7d0c236609be85879ca
00000000 16 FE FD 00 01 00 00 00 00 00 00 00 28
00000000 14 00 00 0C 00 03 00 00 00 00 00 0C 3D 83 B7 D0
00000010 C2 36 60 9B E8 58 79 CA
d: 3d83b7d0c236609be85879ca
v: 3d83b7d0c236609be85879ca
add MAC data: 1400000c000300000000000c3d83b7d0c236609be85879ca
server finished MAC: 1478e52425329b327b305603
nonce: de85c680000100000000000000000000
key: f5954dbf6d310962c0263eeb2ff53622
0001000000000000c1d17e5e2820908cd2a74b43d216e43a7656d659b59291b21636b6290cdd6ba6
00000000 16 FE FD 00 01 00 00 00 00 00 00 00 28
00000000 14 00 00 0C 00 03 00 00 00 00 00 0C 14 78 E5 24

00000010 25 32 9B 32 7B 30 56 03

3.2 Large Scale Monitoring

3.2.1 Problem description

Monitoring is a fundamental part of network management to enhance security. While it is known that guaranteeing a 100 % attack resistant network, even with advanced authentication methods and encryption mechanisms, monitoring helps to track abnormal behaviours, also known as anomalies, coming both from faults (involuntary anomalies) or attacks (voluntary anomalies). While section 2.1 is dedicated in analysing usable security mechanisms, this section is hence dedicated to monitoring which is complementary, *i.e.* both are needed.

In almost every network where security policies have been implemented, there is also network monitoring. This is valid for all kinds of network contexts. In IoT6, three levels are considered as reminded in Figure 9. In addition, all of these levels are affected by the common issue of monitoring: the large data volume to handle.

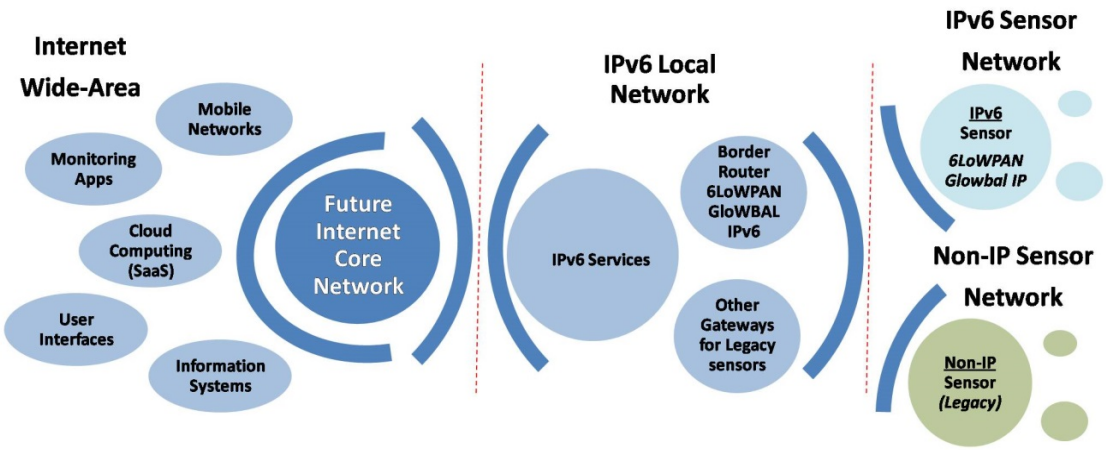

Figure 9: IoT6 Network Structure

In fact, data source for monitoring are various: pure network traffic, logs, alarms of IDS and these different sources provide also various kinds of information (IP addresses, host/domain names, alert level, etc.). To illustrate, sensor networks might be composed of thousands of sensors, which then directly impact the traffic volume on the WSN but also in the backbone network which includes additional traffic as well. Also from an Internet perspective, an operator may have millions of customers communicating outside its own network and so the traffic to analyse is very high. As an example, a major ISP in Luxembourg is able to collect more than 100,000 flows per second nowadays. While collecting is possible, automated or semi-automated analysis is still a major challenge.

3.2.2 Approach Overview

In order to assist the network administrator in tracking anomalies, we propose to extend MaM (Multi-dimensional aggregation Monitoring) [74] to be applicable for IoT6. This tool allows the aggregation of data which are composed of features which the instances can be represented hierarchically. For example, IP addresses, domain names or even geographical coordinates are usable features.

The main advantages of the tool are:

- Aggregation of data into representative trees with a limited size, which thus helps manual analysis and reduces the overhead of an automated analysis.
- The feature space is not divided a priori and so the aggregation granularity varies over the space according to the events being monitored, i.e. reaching a user-defined

volume- based threshold like the number of bytes. For example, the IP addresses are aggregated into different subnetwork sizes and MaM can choose a /50 for a subpart of the IPv6 address space while using /56 for another subpart of IP addresses.

- There is no order relationship between dimensions. Assuming that both source and destination IP addresses are considered, the user usually has to decide what to aggregated first, for example the destination addresses and then the source addresses. MaM alleviates the user to make such a choice. In addition, MaM can decide to aggregate the same feature several times. For example, all traffic coming from a /50 network can be aggregated, then the latter is aggregated regarding the destination IP addresses, and finally again regarding source IP addresses with a prefix higher than 50.

This process is illustrated in Figure 10.

As highlighted in the previous sections, MaM only handles features which can be represented in a hierarchical manner such as IP addresses or domain names.

While only IPv4 addresses were supported, the support of IPv6 addresses was added in the context of IoT6. Geographical positions are also an interesting feature in the context of IoT that MaM is able to handle it.

Besides, we also propose some metrics in order to automatically detect anomalies in Netflow records [39] but also in DNS traffic as this is also a possible choice of a repository in IoT6.

In next sections, MaM is briefly summarized; then IPv6 support analysis and implementation is detailed before highlighting use cases with an automated analysis of real data.

3.2.3 MaM

MaM performs multidimensional aggregation using an underneath tree structure. The general process is described in Figure 10: MaM processing steps. The objective is to aggregate multidimensional data into a single tree structure for each time window. For being executed, the user has to provide input data as well as a data model describing the type of data and how to parse and aggregate it. Therefore, the first step is to parse the input data. For each data instance, usually a line in a file, MAM creates a node accordingly. The latter is inserted into the current tree based on the hierarchy of each dimension (next sections provide details). These steps are repeated until the tree has to be compressed.

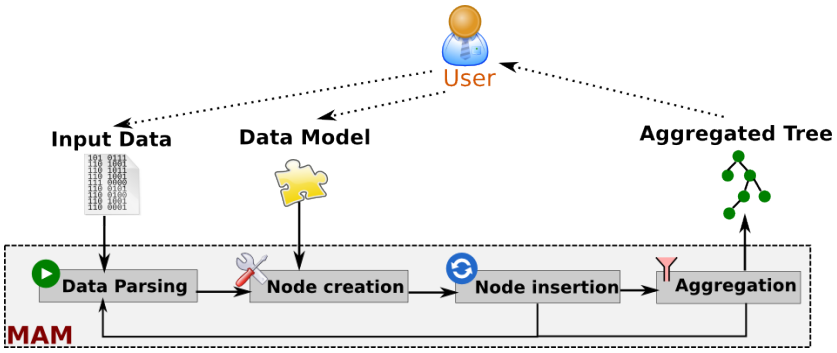

Figure 10: MaM processing steps

Every node is composed by a set of instances of every dimension (feature) and an accumulated value representing the portion of traffic for those combined dimension instances. An example from the original paper using IPv4 addresses is presented in Figure 11.

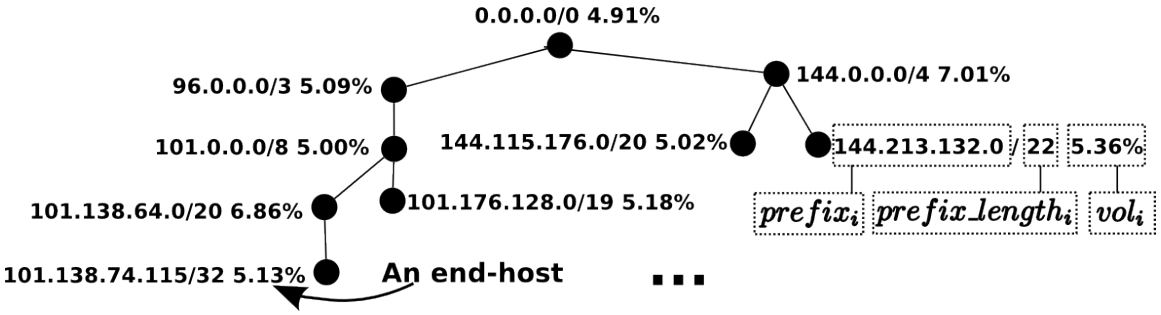

Figure 11: Aggregation on IPv4 traffic

Aggregation is done for multiple dimensions over time and space, by using two parameters: a user-defined threshold α indicates that those nodes containing an accumulated value of the traffic > α (number of bytes in percentage) are kept and a window of η seconds defining a size of a time window in seconds. Using those two parameters, a tree is constructed and the number of nodes is kept under a user defined value called MAX_NODES. To keep the tree size under that value we implemented two strategies. If the timing information is not provided, MAM can be still used to create a single tree for the whole dataset.

- Root aggregation: this strategy performs an aggregation from the leaves to the root through a post-order traversal. For each visited node, if the aggregated volume is higher that the aggregation threshold, α, it is aggregated to its parent node. If after this process, the number of nodes in the tree is still higher MAX_NODEs, α is artificially increased by a factor, by default 2 until the number of nodes is acceptable. Least Recently Used (LRU): in this case the least recently used nodes are aggregated into their parent. In fact, each node age is kept as additional information and the age is updated when a node is inserted or updated as well as all its ancestor nodes.

3.2.4 IPv6 Support

MaM does not support IPv6 by default. Therefore, we have extended MaM in such a way. The CIDR (Classless Inter-domain routing) was initially proposed in IPv4 for aggregating IP subnets into larger ones such that the number of routing table entries remains lows. To do so, two addresses or subnets are aggregated into a single subnet by considering the most common prefix.

Obviously, the same approach is valid for IPv6 as highlighted in [38]. As an example, the subnets 2001:0C00::/48 and 2001:0C00:FFFF::/48 can be aggregated into the common subnet 2001:0C00::/32.

IPv6 addresses are composed of 128 bits but, as agreed by IETF, only the first 64 bytes should be used to characterize the network while the remaining 64 bytes are used as an interface identifier. This basically means that aggregation should be limited to the 64 first bits (high order) in normal cases.

The global unicast addresses format is as follows:

- bits 1-48: routing prefix
- bits 49-64: subnet id
- bits 65-128: identifier

The subnet id might be even smaller but to keep a generic case, we consider that aggregation will perform on the first 64 bytes as the routing prefix combined with the subnet id compose the complete network prefix used for achieving the end-to-end packet routing.

There are also other types of addresses, like the multicast addresses or link local addresses.

Since they have a fixed prefixes, the corresponding addresses will be automatically aggregated into a common subpart of the tree generated by MaM. However, as the goal of MaM is to perform large scale monitoring, the tool fits better traffic analysis at a large scale and so is better suitable for unicast addresses. For example, attackers may target hosts through Internet and so through their unicast addresses. More details about IPv6 addressing are available in [38] and D2.3.

In MaM, each feature is defined such that two instance of data, a and b, can be compared if known that:

- a is an ancestor of b,
- or b is an ancestor of a,
- or a and b are disjoint.

For instance, 2001:0C00::/32 is an ancestor of 2001:0C00::/48 while 2001:0C00::/48 and 2001:0C00:FFFF::/48 are disjoint.

The parent is the direct ancestor of an instance. For example 2001:0C00:0C00:/47 is the parent of 2001:0C00::/48 and 2001:0C00:FFFF::/48. Regarding IPv6 addresses, a parent can have only two children depending on the next bit after the prefix, 0 for 2001:0C00::/48 and 1 for 2001:0C00:FFFF::/48. Actually, the parent is the common prefix of its two children.

These notions are highly important for MaM as the tree construction is based on such hierarchical relationships. As this construction is independent from data, the implementation has only to take care of some functions for manipulating the hierarchical relationships.

MaM is implemented in python. Adding support to IPv6 required the development of a new class which inherits from *Dimension.* As the IPv6 addresses may be represented hierarchically similarly to IPv4, the new implemented class inherits directly from IPv4 with some extensions, in particular to handle 128 bits and to consider the limitation the network size to the first 64 bits. Finally, the main functions which have been implemented are the following ones:

- *Direction*: assuming two IPv6 subnets, a and one of its ancestor b, this gives the next bit value of the network b (in the IPv6 addresses) regarding the prefix a. This was illustrated in the previous section
- *Common_prefix*: assuming two subnets, a and b, this function returns the common ancestor

Some additional functions, mainly for visualization purposes, have also been implemented. In particular, graph generated by MaM highlights potentially abnormal facts in an automated manner to guide the network administrator. This is based on the stability of data which is detailed in section 2.2.6. However, as an example, Figure 11 shows a tree which has been outputted by MaM on a sample of IPv6 traffic, potential anomalies are marked in red. The intensity of the colour increases regarding the probability of the node being representative of an anomaly. The size of the nodes is proportional to the proportion of traffic it represents.

3.2.5 Requirements

MaM has been updated with the new features and is available as an open source tool on github: https://github.com/jfrancois/mam

Running MaM requires python 2.7 or later as well as some additional packages:

- python-levenshtein
- python-pygraph
- python-pygraphviz

- Libgv-python
- py-editdist

The main script is *main.py* which launches a graphical user interface. The tool can be used in console by launching multiaggregator.py which allows setting all options. The latter are described in the EXAMPLE file along some examples.

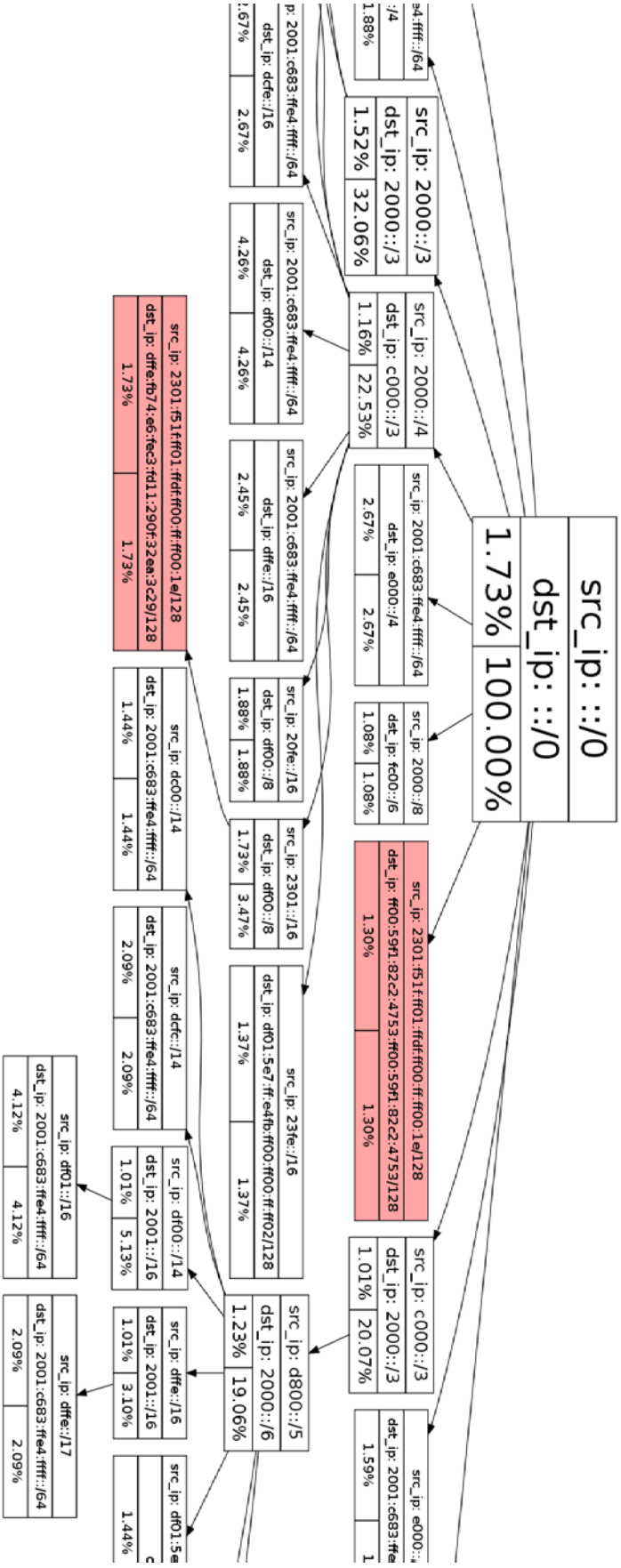

Figure 12: Sample MaM Tree

3.2.6 Automated Analysis

3.2.6.1 Netflow

Flow level traffic information summarizes network traffic by discarding payload contrary to full packet capture. In fact, all packets sharing common properties (mainly IP addresses, protocol and ports) and sending within a restricted time interval are grouped together to form a flow by aggregating number of packets or bytes and marking the flow with a timestamp. There are different versions developed where NetFlow [39] is the one led by Cisco. Netflow or other flow-based approaches are now considered as a standard for IP monitoring.

For our test, real Netflow captures were provided by one major ISP in Luxembourg. As assumed to be free of attacks, we also inject a realist attack in the same manner as in [40] for assessing the ability of our approach to catch valuable information about anomalies.

The duration of the capture is 26 days with an average number of flows around 60,000/sec. A total of 279815 unique IP addresses using 64470 different UDP and TCP ports are represented.

To evaluate the aggregation helpfulness in a malicious environment, a DDoS attack has been injected against web servers running on TCP port 80 with a repeated sequence (3 packets and 128 bytes) sent by burst of 10 packets every 60-120 milliseconds. The attack is repeated every two minutes.

MaM is configured with η = 25s, α = 0.05 and then outputs trees sequentially. Since traffic changes during the attack, the same is expected with trees. As MaM is using tree based data structure, there exist plenty of tools and metrics from state of the art than can be used under a small adaptation. Thus, to observe changes in the tree, the edit distance between two consecutive trees can be used.

The Levenshtein distance [42,41] is usually refereed as edit distance between two strings. It is defined as the minimum amount of operations to transform one string into the other (deletion, insertion and relabeling or substitution). As an example, the strings ”sos” and ”sbs” have a distance of 1 by performing one substitution of ”o” into ”b”. This notion of distance was initially defined for strings where every operation has a single cost of 1. This can also be applied to the nodes of MaM trees. As a node is a more complex structure, thus deletions and insertions are always associated to a cost of 1. However substitutions in multidimensional nodes are decomposed into each dimension. According to the formal definition of a M multidimensional tree, a node was defined as $n_i = < \{f_{i,1}, \dots, f_{i,m}\}, vol\_i\}$ where each $f_{i,j}$ represents the value of the node I regarding one dimension, for example the source IP address. $vol_i$is the volume corresponding to this node, usually the percentage of bytes of packets that this particular nodes represents.

A distance function per each dimension can calculate the cost of an operation as between the *ith* and the *jth* tree as:

$$\sum_{k=1}^{m} distance_k(f_{i,k}, f_{j,k})$$

Thus, the distance function has to be defined for each dimension. In our case, we define its based on the longest common prefix between two IP addresses.

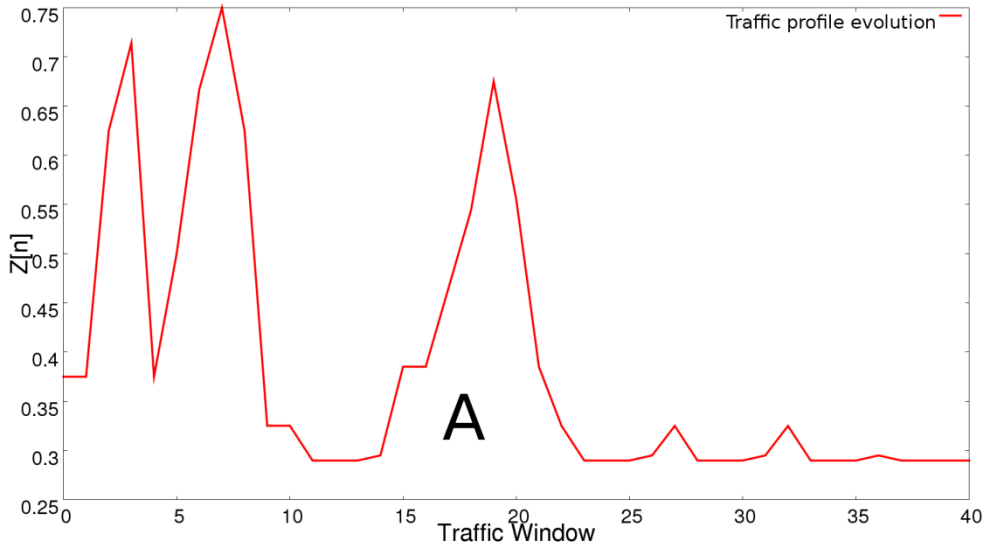

Figure 13: Edit distance under a Denial-of-Service attack

3.2.6.2 DNS

The DNS (Domain Name System) [Moc87] is a major player in Internet nowadays. Its main objective is to perform the mapping between a human readable name like *www.uni.lu* and an IP address. It is also possible to easily reallocate resources transparently by changing this mapping on the fly. As service location is also essential in IoT. As this is a well-established protocol, IoT6 also envisions its usage in D2.3 for mapping service name to IPv6 addresses. Besides, such a standard protocol will helps the integration of IoT6 components in the current Internet.

However, DNS is also targeted by attackers which the main goal is to perform a denial of service attack in order to make the service working improperly. In particular, the objective is to prevent the users from finding the right IP addresses corresponding to the service they look for and in worst case, redirect them transparently to attacker-owned services [Ale10]. This is also known as DNS spoofing, for which we can expect that accelerating the deployment of DNSSec [66] will help to fight it efficiently.

Nevertheless, there exist other threats which cannot be mitigated using DNSSec due to its design. Especially, botnets [67] also relies on DNS for empowering their robustness using fast-flux techniques[68]. Fast-flux is used to support various kinds of malicious activities (DDoS attacks, spam, espionage, etc.) by updating regularly the hosting of some offensive content pointed by the same unique DNS name. It can be also be used by the bots to locate the coordination point (the Command and Control channel). For improving their disruption avoidance, malware can compute algorithmically and regularly the DNS names to look for. Phising is also an effective attack by attracting users using specific keywords like brand names or attractive keywords [69, 70]. For example in the context of IoT, an attacker could register a domain like *securelightservice.unilux.lu* for which the user would imagine access to the lights management service of the University of Luxembourg while *unilux* has no relationships with the real domain name *uni.lu*.

Hence, DNS monitoring is necessary to track malicious or attacked domain names. We propose in this section to enhance traditional DNS monitoring using MaM. Notably, it has been observed that malicious domain names are requested during short life time while the associated IP addresses to a single domain varies in high proportions from subnets located at various locations in the IP address space. This is in particular highly visible for fast-flux based botnets [71], Hence, our method consists in evaluating the steadiness of the association

between a domain name and the corresponding IP addresses. Since these features can be represented in a hierarchical manner, MaM is a natural candidate for analyzing large scale passive DNS dataset [72]. The window time based aggregation of MaM allows assessing the dynamic of the DNS mapping. This work will be also published in [73].

Assuming that a service provider collects the results of their users' DNS request by passive monitoring, MaM is hence leveraged to aggregate the collected records regarding simultaneously the fully qualified domain names (FQDN) and the IP addresses. An example is shown in Figure 14, where each node is represented as a FQDN, an IP subnet, the percentage of corresponding request (*vol*) and the percentage of corresponding request including its children (*acc_vol*).

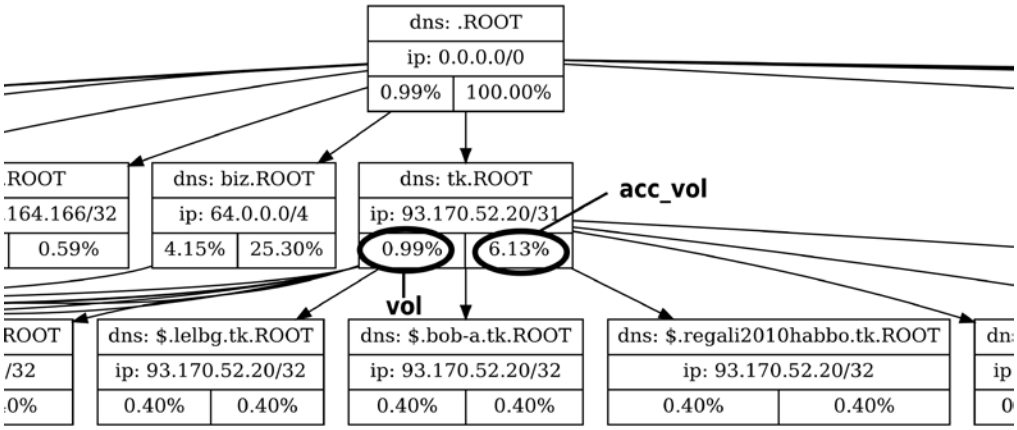

Figure 14: Sample tree for IP Address and FQDN

For evaluating the steadiness of the mapping between domain names and IP subnets, we proposed the following a model:

- A sequence of N tree $S = \{T_1, T_2, \dots, T_N\}$
- Each node $n \in N$ is defined as:
  - n_{acc_vol}, the cumulated proportion of activity volumen as defined in MAM,
  - n_{prefix}, the IP prefix,
  - n_{prefix_len}, the IP prefix size,
  - n_{dns}, the set of names in the FQNS like for instance *{www, uni, lu}* for www.uni.lu.

To compute the steadiness of a domain name, the intuitive idea is to check if the same domain in previous time windows was already associated to the same subnets. MaM has been chosen for its scalability but aggregation may lead to information loss. Hence, finding exactly the same domain is often impossible. Therefore, we adopt a slightly different approach where the objective is to find the most similar domain and generally the most similar node (*mostsim*). To do so, we define the notion of similarity regarding the hierarchical structure. In fact, two nodes have a non zero similarity if they represent overlapped spaces in both dimensions (DNS and IP). The similarity is higher when the node is more specific (higher prefix size, more levels in domain names). To find the node n_2 which is the most similar to n_1, MaM offers its standard construction mechanism to traverse the tree of n_2 Therefore, this is similar to find the right position for inserting n_1 in the tree, whih corresponds to the most specific location.

Assuming two nodes *n1*and $n2$, the similarity is defined as:

$$sim(n1, n2) = \frac{1}{3} IP_{sim}(n1, n2) + \frac{1}{3} DNS_{sim}(n1, n2) + \frac{1}{3} vol_{sim}(n1, n2)$$

where:

$$IP_{sim}(n1, n2) = 1 - \frac{\left|n1_{prefix_{len}} - n2_{prefix_{len}}\right|}{128}$$

$$DNS_{sim}(n1, n2) = \frac{|n1_{dns} \cap n2_{dns}|}{|n1_{dns} \cup n2_{dns}|}$$

$$vol_{sim}(n1, n2) = 1 - |n1_{accvol} - n2_{accvol}|$$

Therefore, the similarity is a value bounded between 0 and 1. The steadiness of the node *n1* in T_i is evaluated regarding the similarity of n1 to the most similar node in the previous tree, T_{i-1} and the steadiness of the latter.

$$n1 \in T_i, n2 \in T_{i-1}, n2 = mostsim(n1, T_{i-1})$$

$$stead(n1) = sim(n1, n2) + \mu\, stead(n2)$$

To globally assess the steadiness, we define the global persistence as:

$$pers(T_i) = \frac{\sum_{n\ in\ T_i} stead(n)}{|n\ \in T_i|}$$

To evaluate our proposal, we collect passively DNS traffic from real users. However, since IPv6 deployment is rather limited, most of requests are of type A (IPv4 addresses). For the sake of evaluation, we use such records since it does not impact the core of the approach. The only needed adaptation is to change the denominator to 32 in the definition of IP_{sim}.

The ground truth is then given using well known domain blacklists:

- MalwareDomain : http://www.malwaredomains.com/
- Exposure http://exposure.iseclab.org/
- FIRE: www.maliciousnetworks.org/

Thus, two datasets can be constructed:

- 661968 benign domains corresponding to 164559 unique addresses
- 173066 malicious domains corresponding to 174619 unique addresses.

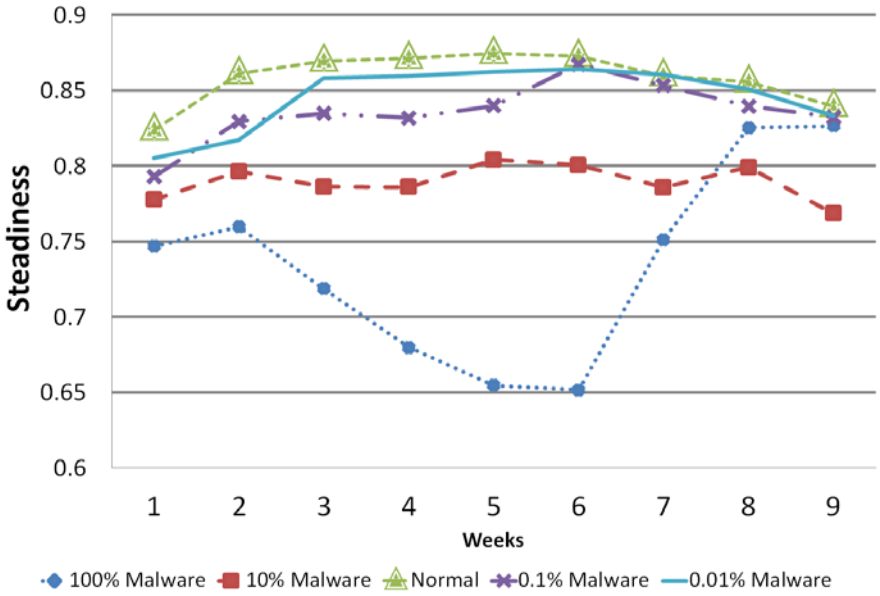

Figure 15: Comparison of average steadiness for 10 weeks period

In Figure 15, the global persistence or average steadiness is measured during ten consecutives weeks which corresponds to the greatest number of records. For this research, we were interested in observing how steadiness varies when different percentages of malicious domains are present. Thus data sets were generated by injecting records of malicious domains into the normal dataset. In this way, it was possible to create datasets with various proportions still respecting the timing information. To compare results to a baseline and contrast the variation in steadiness, the steadiness evolutions for purely malicious and for normal domains were calculated. The corresponding steadiness values are separable in Figure 15. When both malicious and normal domains are mixed together, steadiness is impacted and the values are logically lower than when only the normal dataset is used. For instance, even only 0.1% of injected malicious domains decrease the steadiness value.

Therefore, our approach is able to detect when an anomaly occurs. However it is also important to detect which domains are responsible for, *i.e.* the malicious domains. In the next experiment, we assume that nodes in the tree having a steadiness lower than a threshold are assumed to be malicious. By varying such a threshold, the true positive rate and false positive rate logically varies. Figure 16 shows that a threshold set to 0.5 is able to detect 73% of malicious domains with very few false positives (1.6%).

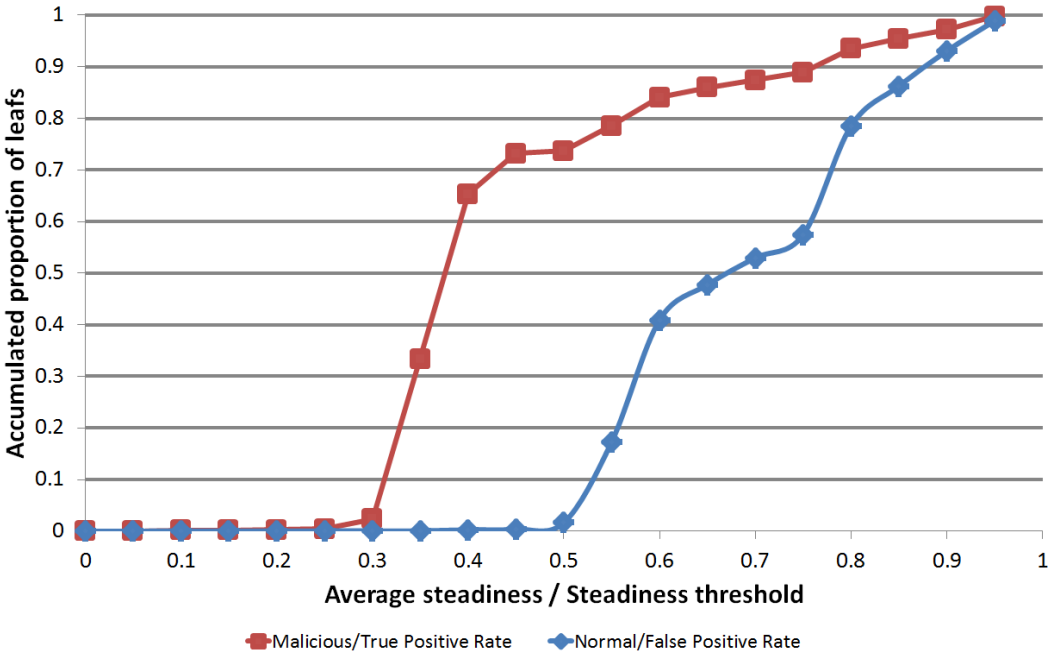

Figure 16: True and false positive rates assuming that malicious nodes have a steadiness lower than the threshold (x axis)

While 1.6% may represent too high a number of false positives, 0.45 is a better value according as the same true positive rate (73%) can be achieved with only 0.3% of false positives. This represents a highly acceptable result.

3.2.7 Outcome

We have adapted MaM to be used with IPv6 traffic because monitoring is a vital task for a network administrator. As shown, it includes various additional features like domain names, service name or even GPS coordinates. All of them are relevant in the context of IoT and especially in IoT6. Therefore, we have demonstrated how to use such a tool in a different context through a manual or automated analysis.

3.3 Content Centric Networking Security

As a research project, IOT6 will envision both short term and long term aspects of IoT communications. If the major part of our work will be dedicated to the improvement of current technologies to achieve our goals of secure and pervasive architecture for IoT, another part of our work is to think more deeply about how communications could be redesigned to enable the requested security and pervasive features by design.

In this context, a new communication paradigm, Content-Centric Networking (CCN), has recently emerged and has strong arguments for being used for IoT communications. In fact, it provides communications with both security and pervasiveness properties by design while these features are hard to achieve within the IP world.

3.3.1 Introduction to CCN

Developed at PARC by Van Jacobson and his team [43], CCN is also known as Named Data Networking at a larger scale [44]. Building on the observation that today's communications are more oriented towards content retrieval (web, P2P, etc.) than point-to-point communications (VoIP, IM, etc.), CCN proposes a radical revision of the Internet architecture switching from named hosts (TCP/IP protocols) to named data to best match its current usage. In a nutshell, content is addressable, routable, self-sufficient and authenticated, while locations no longer matter. Data is seen and identified directly by a routable name instead of a location (the address of the server). Consequently, data is directly requested at the network level (not its holder, no need of DNS). To improve content diffusion, CCN relies on close data storage because storage is proven cheaper than bandwidth: every content — particularly popular one — can be replicated and stored on any CCN node, even untrustworthy. People looking for particular content can securely retrieve it in a P2P-way from the best locations available.

CCN has two main types of packets, *Interest* and *Data* as seen in Figure 17. A user who wants to access a given content sends out an *Interest* packet, specifying the name of the content (as defined by CCN Nomenclature Content Name) to all its available faces. Faces can be anything which can serve as medium for transmitting and receiving data. A node which receives this packet and that can 'satisfy' the *Interest* then sends out the corresponding *Data* packet onto the face from which it received the *Interest*. By definition, CCN nodes are stateful and only forward *Data* on the back path if an *Interest* was emitted beforehand.

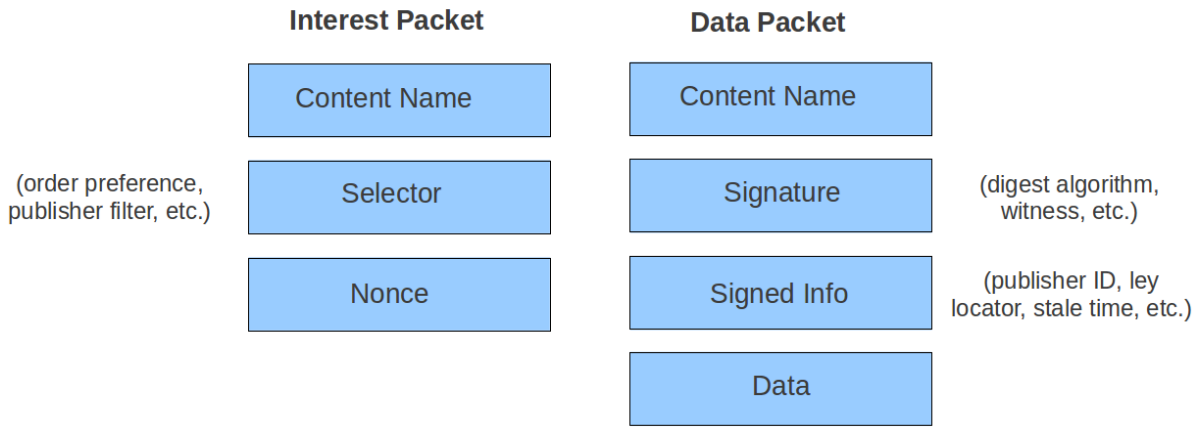

Figure 17: CCN packet structures

Data can only 'satisfy' a specific *Interest* if the ContentName of *Interest* packet is a prefix of the *Data* packet. CCN names are defined in [43] as *"opaque, binary objects composed of an*

(explicitly specified) number of components". This structure allows a fast and efficient prefix-based lookup similar to the IP lookup currently used. It also allows names to be context dependent *i.e. /ThisRoom/Printer* references a printer in the current room. This context-naming could make possible efficient context-aware service discovery in the future Internet of Things.

CCN nodes are composed of three main table structures, presented in Figure 18: CCN forwarding engine - which handle the forwarding of packets. At the arrival of an *Interest* packet on any given face, the engine performs a longest-match lookup on its structures and action is taken depending on the lookup result. The first structure to be searched is the Content Store. It can be seen as a buffer memory of past *Data* packets on the current router. IP routers also have such a buffer but it is purged once the packet is forwarded. The Content Store however preserves the *Data* packet based on LRU scheme and enables therefore a fast retrieval of currently popular demands. If there is a match, the router forwards its local copy of the content to the face on which it received the *Interest* and updates its Content Store accordingly.

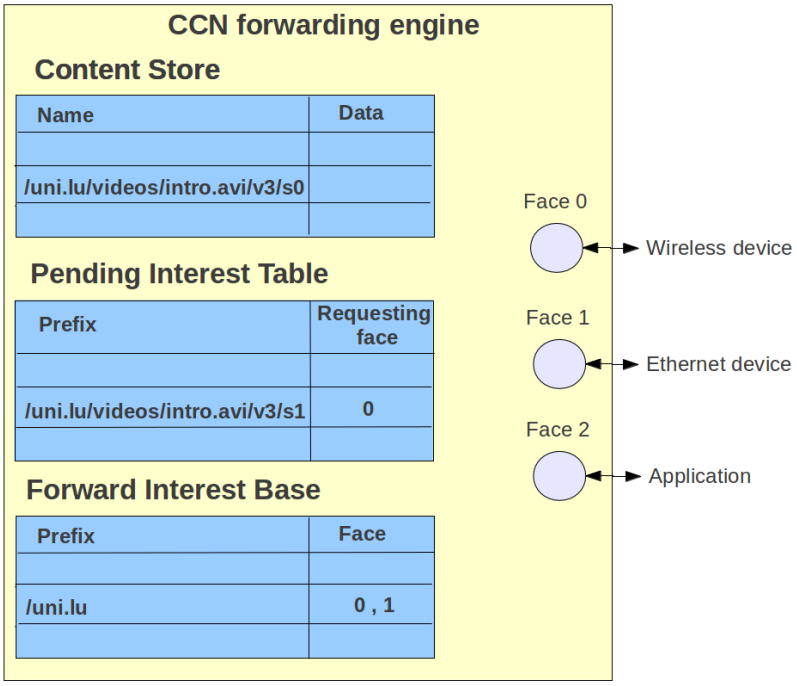

Figure 18: CCN forwarding engine

If there is no match in the Content Store, the lookup is launched on the next structure which is the PIT. The PIT stands for Pending Interest Table and keeps record of *Interests* waiting to be resolved upstream by other content source(s). If a received *Interest* matches an entry in the PIT, the engine compares the faces recorded for that entry. If there is already one existing, no update is made. Otherwise, the face form which the *Interest* was emitted is simply added to the list of already waiting faces.

If no match-up is found in the PIT then the engine searches in its last structure: the FIB. The Forward Information Base keeps record of potential content source(s) and works similarly to its IP counterpart except that it stores a list of possible providers for a given name rather than a single one only. If a match is found, the engine then creates a PIT entry for the given *Interest* and it is forwarded to all faces specified in the FIB entry. If no match could be made, it means that the current router has no information on the demanded content and discards the *Interest*.

CCN has also built-in strategy and security layers. The strategy layer is used to define policies to select which face is the best for given contents. In fact, due to its design, FIB entries

contain multiple faces. CCN can send periodically *Interests* to all outgoing faces without fearing of loops and thereby testing which of the faces responds the fastest. This one will be used as preferred until another round of this experiment yields to a different result. Criteria for experimentation interval can be a threshold of packets sent, a time out, change of the SSID, etc.

The security layer ensures that the content received by a previously announced *Interest* is authentic. As in CCN only the content matters but not the route it takes, the only thing which needs to be checked for authenticity, consistency and integrity is the content itself which reversely means that end-to-end encryption is not needed any more. Key management is another issue often discussed. In [43], several solutions are discussed which range from a PKI to PGP like web-of-trust.

3.3.1.1 Pervasiveness by design

This new routing paradigm is based on a plain-text hierarchical naming instead of regular hosts' IP addresses so that names are directly intuitive and do not need an indirection mechanism between names and contents (no need of DNS, DHT). An example of this hierarchical naming structure is presented in Figure 19 for a content named "ccnx:/uni.lu/videos/intro.avi"

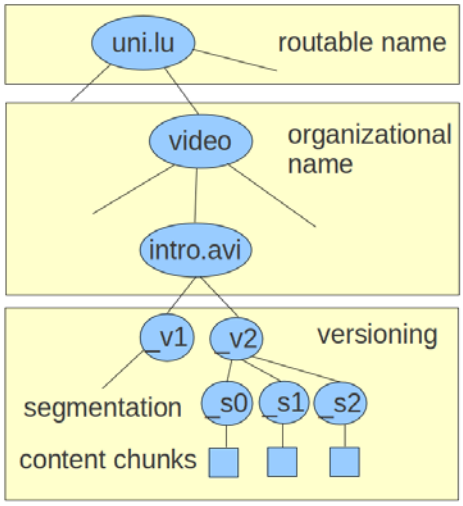

Figure 19: Hierarchical naming of CCN content

REQ ccnx:/nytimes.com/today ->
<- RESP ccnx:/nytimes.com/today
ccnx:/nytimes.com/20120406/index.html
<NameMAC 3fde... />
<DataMAC 07a2... />
<html>
...
<html/>

This introduces a second property which is of great interest for our work in contextual networking: this addressing scheme and name space introduce the possibility of context-aware resolution of content. Using the words of Jacobson et al in [43]; *"Name prefixes may also be context-dependent such as /ThisRoom/projector to exchange information with the display projector in the current room or Local/Friends to exchange information with any friends in the local (broadcast) environment."* This allows pervasive services using CCN connectivity to use a context-aware name space and benefit from a native support of context-awareness, in

particular to enhance service discovery.

CCN has several interesting features for the IoT beyond the native support of context-awareness and service discovery due to the name space,. As no IP address is strictly bound the network interface on a CCN device, CCN can use opportunistic routing and request *Interests* to every network interface available at proximity at a given time (for example, a close wifi-capable device). This multiple and parallel network interface usage drastically increase the connectivity of mobile nodes with the possibility to automatically failover when a face become unavailable. An application that can full benefit from these features is for instance the case of tactical and emergency MANETs [50]. Moreover, these features can be experimented directly on mobile devices thanks to the Android implementation provided by CCNx [48] and the Contiki implementation provided by INRIA Madynes [49].

3.3.1.2 Security by design

Content Centric Networking improves the security of Internet communications in many ways. First of all, CCN messages cannot be sent toward a node without any prior *Interest* request from that node (*Data* cannot be pushed, only pulled) which makes the classical denial of service scheme inefficient as the attacker would first need his target to generate a lot of *Interests* to enable the DoS attack.

The paradigm shift of CCN makes every node capable to answer a *Data* request. To ensure the security of communications all data is authenticated, contrary to the connections it traverses. The security scheme should provide better results. For example, a secure connection to a mail server does not avoid SPAM mails to be received while with CCN, SPAM mails will fail the authentication as untrustworthy *Data* and will be discarded. So, CCN strongly relies on cryptography to authenticate the contents so that users can clearly know who emitted the content and can discard those from untrustworthy sources to avoid malware. Also, encryption is used to ensure privacy.

CCN provides native security and privacy by encryption with lower overhead than current protocols [43]. To securely authenticate content, CCN has to bind the content name, the content itself and the content provider. To do so, the following information is embedded in each CCN data packet:

Signature(Name,Content,SignInfo)
SignInfo includes: cryptographic digest or fingerprint of publisher's key, key or key location

In [47], Smetters et al. propose the following description: a new content creates a mapping triple: $M_{(N,P,C)} = (N, C, Sign\_P(N, C))$ where M is the Mapping, N the Name, C the Content. With every data must be provided a way to retrieve the key of publisher and mapping evidence.
Content can be authenticated by every node using public key signatures and different signature algorithms are available to find a trade-off between the needed security and performances. The key-stone of CCN security is the trust in the publisher. To ease key management in CCN, Jacobson et al. propose to see an organization as a content name and public key as a Data. They propose to use the SDSI/SPKI where keys are mapped to identities via namespaces (CCN names) so that there is no single source of trust like the current certification authorities. This scheme performs security by allowing the traceability of data.

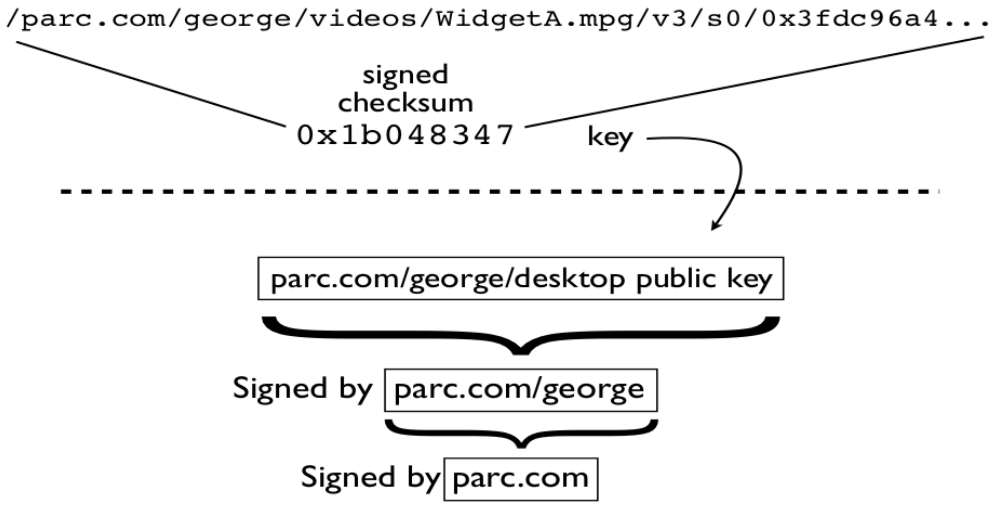

Figure 20: Trust management by associating CCN names with publisher keys [43]

If CCN improves security in some points, it also raises the possibility of new kinds of attacks. For example, unlike terminal hosts which are less exposed to attacks, CCN routers are more vulnerable than IP routers because of their stateful nature and the complexity of the management of their inner tables from which result their performance and quality of service. All these new algorithms and their implementations need to be assessed from a security point of view before being trustable.

3.3.1.3 Security limitations

However, so far, there has been little investigation of the security of CCNs. We currently lack the knowledge of possible security issues related to CCN and particularly: 1) how could malware benefit from CCN architecture to improve its diffusion; and 2) how could malware attack CCN in order to intercept or disrupt the communications carried? We have arguments indicating that the first question may become critical. In fact, one of the key elements of CCN architecture is the replication of content and their diffusion from the closest available copy. This means that no direct connection to the initial source is needed in many cases, like in a P2P network. This feature is thus very attractive from a performance and a scalability point of view, but will also make the tracking of malware very difficult as they can be replicated and diffused from potentially any network node! When malefactors will leverage CCN, it will become almost impossible for a centralized entity to monitor and control the diffusion of tracked content over the network after its initial release. Although there is currently no malware that leverages CCN, the history of the Slapper worm shows that malware creators can and will embrace state-of-the-art academic research approach if a return on investment is possible.

Therefore, new methods and tools to monitor and revoke contents at the scale of the CCN networks should be designed. Another argument is based on the work from DiBenedetto et al. [45]. They propose an application over CCN that enables privacy-preserving communications while introducing less relative overhead than TOR running over IP. This could lead to malware hiding C&C channels in anonymous communication using CCN. Beyond the question of the CCN paradigm that can support malware, the second critical problem is to assess whether the CCN architecture and algorithms are robust enough to resist attacks. In particular, due to stateful routers (as the route between a content and the requester has to be memorized) network devices are exposed to new threats that can be exploited for attacks. Attackers may disturb the management of CCN nodes' tables (Pending Interest Table, Forward Information Base, Content Store Table) from which result their performance and quality of service. This can lead to new possible DoS (Denial of Service) attacks or to

malicious monitoring, that must be detected and mitigated. In fact, Tobias Lauinger [46] identifies several attacks related to caches, in particular denial-of-service attacks against CCN routers, but he only investigates one of these, “cache snooping” that enables attackers to efficiently monitor the content retrieved by their direct neighbours. Smetters et al. [47] of PARC propose authenticating the links between names and content, in addition to names and content themselves, in order to identify trustworthy content and avoid malware diffusion. However, the necessary PKI is hard to set up and if a provider becomes malicious, revocation remains a major problem that must be addressed in future work.

To conclude, security by design may not be sufficient without proper tools that will enforce the security policies applied in a CCN network. Among others, CCN currently lacks:

1) A monitoring architecture that can detect suspicious traffic by analysing the activity of CCN nodes tables
2) A firewall that can apply the security policy of a network (discard untrustworthy content and provider) based on name prefix or signature

3) A distributed key management system to enforce an efficient and safe authentication of CCN content and content providers as well as the revocation of malicious ones

3.3.2 CCN monitoring architecture

3.3.2.1 Introduction

From a management point of view, Content-Centric Networking introduces new challenges. Firstly, it is hard for a content provider to monitor and control the diffusion of its content over the network after the initial release which leads to accountability issues because content can be distributed from any CCN node without requesting the original provider. While it is an important quest from a management point of view, focus is made on the security aspects which present a more critical drawback of early deployment of CCN. Secondly, CCN routers are stateful as the route between content and the requester has to be memorized. This stateful nature can lead to new possible DoS (Denial of Service) attacks which exploit the CCN routers limited memory size and these kinds of attacks must be detected and mitigated. In this deliverable, this second issue is addressed because security issues could highly decrease the appeal of the technology and reduce further research efforts. To that end, we (1) model a DoS attacks and (2) propose a Monitoring Architecture for Content-Centric Networking which is able to detect them.

3.3.2.2 Threats description

Content Centric Networking improves the security of Internet communications in many ways. First of all, CCN messages cannot be sent toward a node without any prior Interest request from that node which makes the classical denial of service scheme inefficient as the attacker would need his target to generate a lot of Interests to enable the DoS attack. Also, CCN strongly relies on cryptography to authenticate the contents so that users can clearly know who emitted the content and can discard those from untrustworthy sources in order to avoid malware. If CCN improves security in some points, it also raises the possibility of a new kind of attack. Unlike a terminal host which is less exposed to attacks, CCN routers are more vulnerable than IP routers because of their stateful nature and the management of their inner tables from which their performance and quality of service result. By focusing on CCN routers, new kinds of Denial of Service can be performed. These attacks regarding the tables they target are categorized as follows:

PIT attack: A first attack can be focused on the Pending Interest Table. This table is critical because of the stateful routing of CCN network. If an attacker can manage to fill the PIT with a lot of forged Interests, legitimate Interests might be dropped, resulting in the denial of service of the pending communications. The attack is easy to achieve from the technical aspects. The attacker only needs to generate a lot of Interests whatever is the requested content in order to create entries in its PIT. Such an attack would benefit from distributed attack sources that would make it more difficult to detect. Therefore, the monitoring of the PIT to avoid flooding is a critical point for a safe and efficient CCN infrastructure.

FIB attack: Unlike the IP address space, the CCN address space is not clearly bounded as domains are defined through strings rather than a small IP prefix. Therefore, a possible attack consists in generating and advertising a lot of contents belonging to different domains in order to fill the FIB on a face of the router. In that way, new legitimate domains cannot be routed through the CCN device in which the interface is full. This DoS is critical because one of the major interest of CCN is to allow end-users to directly diffuse their content in a peer-to-peer way instead of relying on big Internet content providers. This attack could reduce the diversity of routable domains and consequently the interest of CCN.

Content Store attack: DoS can also be launched against the Content Store in order to decrease the efficiency of the caching mechanism which is one of the main components that provides the incentive to deploy CCN infrastructures. According to the caching policy, an attacker could generate a lot of download requests for unpopular contents which would modify the distribution of the downloaded contents and update the cache in an inefficient way. From a technical point of view, this attack is hard to achieve for a single attacker as it would need a lot of bandwidth to have a significant impact on the distribution of contents passing through the router.

3.3.2.3 Monitoring Architecture

3.3.2.3.1 Requirements

The monitoring task consists of collecting information about the functioning and the current status of the CCN nodes. The objective is to correctly select and process the necessary information to highlight important facts. In this work, focus is on the detection of anomalies resulting from attacks. As CCN works in a distributed manner with independent nodes, monitoring the network from a global perspective is thus hard. A solution could counter this problem by leveraging a central service interacting with a large set of CCN devices to collect and analyse information. However, this raises serious issues about reliability and scalability and does not fit in the CCN paradigm where each node has to be involved in the functioning of the network including the security related aspects.

Therefore, in order to stick to the CCN paradigm, our architecture is implemented at each CCN device which has to monitor itself for detecting anomalies. As presented in the previous section, a device has three main components: the FIB, the PIT and the Content Store. Each of them plays a crucial role in the well operation of CCN. For example, an abnormal Content Store can provide faulty contents or make the caching inefficient; a badly populated FIB may entail erroneous forwarding and some content may not be accessible any more similar to a bogus PIT which leads to the disruption of the data content transmission over the back path of a request. Therefore, all of these three tables have to be monitored.

The objective of our monitoring architecture is to detect attack patterns by monitoring the recent past activity over the three tables. Since they may contain much information which are related to many actions (lookup, updates, etc.), monitoring and keeping track of all individual

entry or action requires many resources that may delay the process or even affect the entire functioning of a node up to a denial of service in the worst case. To guarantee the scalability and the timeliness, our architecture is designed to represent the three monitored tables by condensed metrics which values can be easily tracked along time.

Finally, devices can also share knowledge for detecting the attacks, in particular for highly distributed ones like DDoS, as well as for preventing future ones or recovering efficiently from anomalies. This may be provided as content where devices can express their interest through the underlying CCN itself. However, as the individual monitoring is already required beforehand, our work focuses on it.

3.3.2.3.2 Instrumentation

As explained in the previous section, there are different components of a CCN node that can be monitored for detecting malicious activities. All of them are impacted and/or impacts the network activity. For example, a node may receive an Interest (ingoing network activity) which has to be forwarded (outgoing network activity), this will update its PIT. Therefore, monitoring the network activity will track the global functioning of a node and its internal components without having a particular monitoring function for each of them.

For detecting attacks, network statistics are retrieved periodically from the CCNx implementation every ***t*** seconds. So, for each time window ***t***, the following metrics are considered for all active faces of the CCN devices:

- $recv\_byt_t$: number of received bytes per second
- $sent\_byt_t$: number of sent bytes per second
- $recv\_data_t$: number of received Data packets per second
- $sent\_data_t$: number of sent Data packets per second
- $recv\_intr_t$: number of received Interests per second
- $sent\_intr_t$: number of sent Interests per second

We also consider more synthetic values on the router status that are also provided by the CCNx implementation:

- on Content statistics: number of $accessioned_t$, $stored_t$, $staled_t$, $sparse_t$, $duplicated_t$ and $sent_t$ contents
- on recent Interest statistics: number of $named_t$, $pending_t$, $propagating_t$ and $noted_t$ Interests
- on total Interest statistics: number of $accepted_t$, $dropped_t$, $sent_t$ and $stuffed_t$ Interests

3.3.2.3.3 Classification algorithm

The objective of the classification algorithm is to label each time window as anomalous or benign. A time window $\mathbf{t_i}$ is a tuple defined as:

$$<recv\_byt_{ti},sent\_byt_{ti},recv\_data_{ti},sent\_data_{ti},$$
$$recv\_intr_{ti},sent\_intr_{ti},\ accessioned_{ti},stored_{ti},$$
$$staled_{ti},sparse_{ti},duplicated_{ti},sent_{ti},named_{ti},$$
$$pending_{ti},propagating_{ti},noted_{ti},accepted_{ti},$$
$$dropped_{ti},sent_{ti},stuffed_{ti}>$$

This work leverages Support Vector Machines (SVM) [58] which are able to efficiently classify data, even if the data points are not separable linearly, while the complexity remains low [60] allowing our solution to detect in real time attacks affecting the monitored CCN nodes. For sake of clarity, we have considered a single attack at a certain time which is handled by 2-class SVM. However, multiple attacks could be detected by building a unique

multi-class classifier [59].

2-class SVM is a supervised method and so requires *M* training samples: Train = $\{(t_1,l_1),\ldots,(t_M,l_M)\}$ with $l_i = 1$ if the time window t_i contains an attack, else *-1*. For enhancing the data separability, these samples are mapped into a higher dimensional space. Defining an efficient mapping function, φ, is a difficult task because it corresponds to add an additional dimensional space over data not given by any features. This however may be avoided by using the kernel function defined as:

$$K(t_i,t_j) = < \varphi(t_i) \,.\, \varphi\,(t_j) >$$

Because our data points are vectors representing the different metrics of a time window, the Radial Basis Function (RBF) is adapted:

$$K(t_i,t_j) = e^{-q|ti - tj|}$$

where *q* is tunable. Different values for q have been tested to choose an optimal which provides the best result.

Once in the space, the points may be linearly separable by a hyperplane which divides the samples of two classes with the maximum margins regarding the hyperplane. This leads to the following optimisation problem.

$$\max \sum_{ti} \alpha_{ti} - 1/2 \sum_{ti,tj} \alpha_i \alpha_j l_i l_j K(t_i,t_j)$$
$$t_i,t_j \textit{ belongs to Train}$$

subject to where $C = 1.0$ determined through initial experiments:

$$\sum_{ti}\alpha_i \; l_i = 0$$

$$\textit{For all } t_i \textit{ belongs to Train}, \; 0 \leq \alpha_i \leq C$$

As highlighted by these equations, the problem solving leads also to determine α_{i} which is used afterwards for classifying a new time window. A major advantage of SVM is that it relies on a subset of initial samples for the decision function, *i.e.* the support vectors which represent the training points such that $\alpha_i \neq 0$. Assuming, t_x, a time window which requires a prediction about the attack, it will be labelled by the following function:

$$f(t_x) = sgn(\sum_{(ti,li)} \alpha_{ti} \, l_i \, K(t_i,t_x) + b)$$

$$(t_i,l_i) \textit{ belongs to Train, } \alpha_i \neq 0$$

3.3.2.4 Experiments

3.3.2.4.1 Attack description and test environment

The most critical threat among the three described is clearly the attack on the PIT table. Attacks on Content Store can just reduce the efficiency of the cache and therefore do not present critical security issues. Also, attacks on the FIB will become critical when large deployments of CCN will occur, with many providers announcing contents, while PIT attacks is already a threat for local deployments involving few CCN nodes. Therefore, the main threat we want to address when monitoring CCN nodes is the Pending Interest Table DoS attack as described previously. To realize the detection, different attack strategies against the PIT in a single attack tool based on the source code of the *ccndsmoketest* [61] program provided in the CCNx implementation was first implemented . The PIT stores Interests according to the faces they belong. To fill the PIT table two dimensions, the number of faces created and the number of Interests requested must be considered.

- **Burst attack**: Sending multiple Interests to multiple faces. Our first strategy sends a given number of Interests on a given number of faces. In extreme scenarios, a great

many Interests to a single face or a single Interest to several faces can be sent. Both dimensions can be combined leading to the following definition: *Attack1(#packets,#faces)* with the aforementioned remarkable values: *Attack1a(1,n), Attack1b(n,1), Attack1c(n,100), Attack1d(100,n)* where *n* defines the attack aggressiveness and may be tuned to study the impact of the attack.

- **Long duration attack**: Keeping alive multiples faces with periodic Interests. Our second strategy consists of making the DoS more efficient by keeping alive a lot of faces with a small number of Interests that are sent periodically. In this case, the attack aggressiveness, *n*, is the number of targeted faces. In this experiment, keep alive Interests are sent every *t=4* seconds.

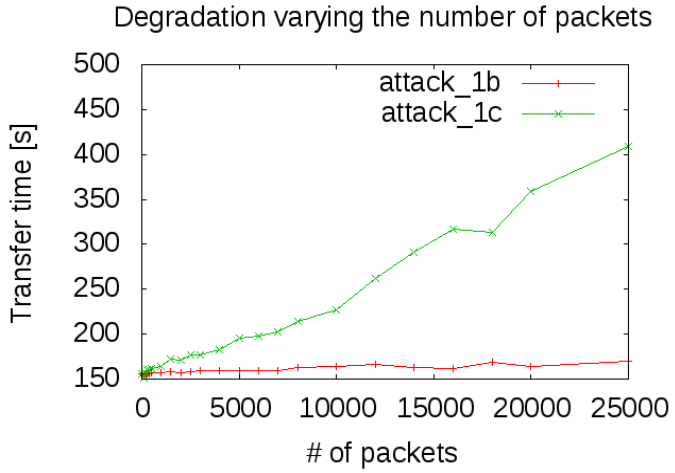

Figure 21: Impact of varying # of packets (attack 1b, 1c)

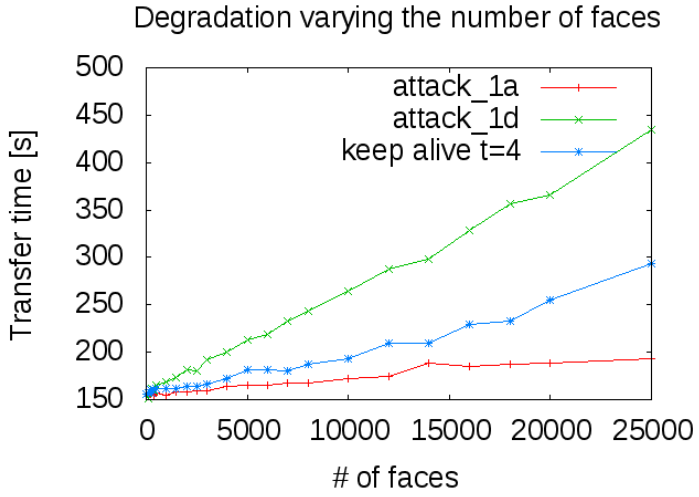

Figure 22: Impact of varying # of faces (attack 1a, 1d, long duration)

We defined the impact factor of our attack as the time overhead introduced when transferring content between our two CCN devices. Figure 23 shows the impact of the number of Interest packets which were injected while targeting a constant number of faces. Firstly, the Interest packet generation was varied in order to inject them over one face (Attack 1b). The same principle was later used but we injected them on 100 faces (Attack 1c). Figure 24 is similar but the number of faces was varied while maintaining a constant number of injected Interest packets (Attack 1a and 1d). Logically, performances are more degraded when the number of faces or Interest packets increase in particular if both are combined (Attack 1c and 1d).

Moreover, multiplying the number of faces used has a similar effect as opposed to sending multiple Interest packets. The second attack strategy of keeping alive many interfaces can also significantly degrade the performance.

3.3.2.4.2 Attack detection

As previously described, attack detection is based on SVM analyzing metrics over time windows. In our evaluation, the size of a window is set to one second. To assess the detection, the following metrics are used:

- True Positive Rate (TPR): proportion of correctly identified windows presenting an attack,
- False Positive Rate (FPR): proportion of windows without attack classified as attacks.

In order to strengthen our evaluation, only one third of the data is used for the training while the remaining is considered for testing and computing the previous metrics. Each experiment is run 10 times including a shuffle of windows for computing the average TPR and FPR. Initial experiments have been done to configure SVM for obtaining a good trade-off between TPR and FPR by using 5000 packet faces respectively as initial data.

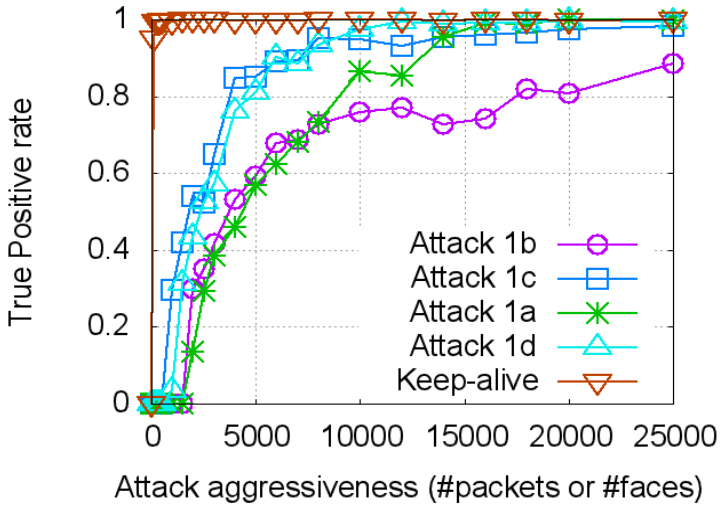

Figure 23: True Positive Rate

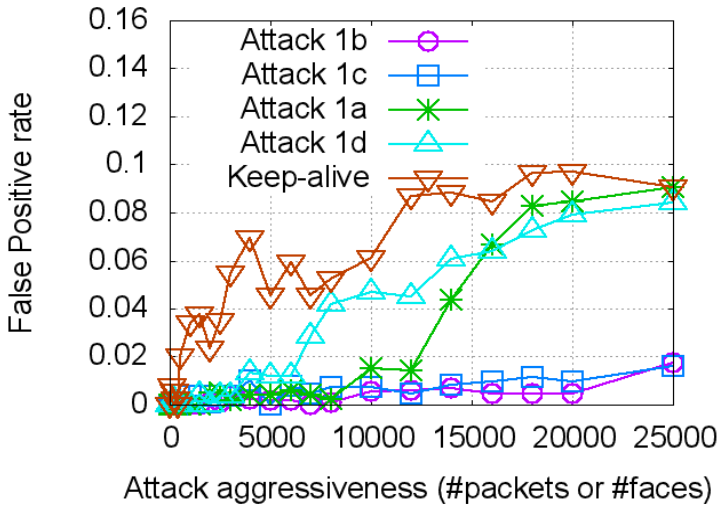

Figure 24: False Positive Rate

In Figure 23 the true positive rate is plotted regarding the attack aggressiveness. This corresponds to the number of Interest packets for attacks 1b and 1c. The latter 1c is detected

easier since the number of faces is multiplied meanwhile by 100 compared to 1b. Once the attack aggressiveness reaches 10,000 Interest packets, the TPR is higher than 95%. Similarly, the Attack 1d is easier to monitor than 1a as the number of sent Interest packets is 100 times higher. Finally, the attack based on keep-alive Interests is well detected in any cases. In fact, such an attack last a longer time and is consequently much more visible.

Figure 24 shows that FPR remains low in most of cases. For the attacks 1b and 1c, generating a lot of Interests, they are never above 2%. The worst values are obtained for attacks involving a lot of faces (Attack 1a, 1d and long duration attack) which seems contradictory as these attacks should be recognized easier when the number of solicited faces increases. In fact, this is due to two biases that we have investigated manually. First, the monitoring interface provided by CCNx gives metrics that are smoothed regarding the time. Hence, the impact of an attack is still visible on the monitoring interface in several time slots once it is finished (slow decrease of certain values over time) implying false positives. This is more noticeable with the keep alive which inject Interest packets periodically. This finding raises the need of a more accurate monitoring of inner values of CCN nodes for security purpose. Second, attacks involving many faces are longer to execute, which leads less windows without attacks. Thus, the training becomes less efficient for normal windows resulting in more false positives.

3.3.2.5 Conclusion

The first monitoring architecture for CCN was presented. While this new paradigm is worth being investigated for the sake of future Internet and IoT communications, it also raises new management challenges which we introduced. Among those, one of the most important problem affecting CCN devices was investigated: the possible denial of service through the flooding of the PIT table. To address this issue, we used the monitoring features of the reference implementation coupled with a classification algorithm based on SVM and which can efficiently detect such attacks with a small computational cost. In fact, we implemented and experimented different attack strategies to perform DoS and all of them can be detected with very low error rates, which could be even lower with a more accurate report of operating values.

Our research opens directions for a great many future works. First of all, our detection mechanism will be implemented within the CCNx libraries in order to enable real-time detection and the usage of associated countermeasure to mitigate attacks. We will then extend our monitoring architecture to monitor the other tables (FIB and Content Store tables) to detect other types of attacks. Finally, we want to extend our test-bed and generate more realistic traces including traffic from different applications. Attack detection was the focus point of our approach and we will also focus on attack prevention in our future work.

3.3.3 CCN firewall

3.3.3.1 Introduction

Despite the fact security aspects have been considered in the design of CCN, which includes, for example, the authentication of contents, several key components that secure the current Internet are still missing in CCN, in particular a firewall able to enforce security policies. In IP networks, firewalls are usually the first level of protection by enforcing the security policy defined by the administrator within the company's network. We propose in this section a comprehensive study of CCN security requirements from which a first CCN compliant firewall is designed, including the syntax and definition of rules. In particular, based on CCN features, our firewall can filter packets regarding both their authentication and the semantic of the content name. We then provide a performance evaluation of our prototype.

3.3.3.2 Firewall design

3.3.3.2.1 Use cases analysis

Regarding the basic functionalities of current firewalls like the open-source iptables [79], the main parameters taken in consideration when writing the filtering rules are the IP address of the hosts, the communication port (linked to the application) and the status of the connection (new, established, etc.). However, as was explained in the previous section that such concepts are no longer available in the CCN world, making the design of a firewall challenging. Despite interesting security features, CCN still needs to provide a way for network administrators to properly enforce their security policy.

To drive the design of our firewall and the requirements of the language, a list of use cases describing the different security considerations that an administrator of a Content Centric Network may encounter is first presented. We first present some general use cases addressed by an IP firewall with analyses in regards to their applicability to the CCN world. CCN specific considerations are then introduced before defining the needed features of our firewall.

3.3.3.2.2 IP firewall general use cases

One of the most important use case addressed by firewalls is the filtering of protocols according to the network policy (*IP_UC1*). A CCN-firewall should be able to filter communications on the same principle with the ability to differentiate the different kinds of traffic (web, mail, voip, p2p, etc.). For example, web traffic (http) may be authorized in the network while other protocols (ftp, p2p, etc.) are forbidden. A second crucial point of IP firewall is to filter traffic according to the status of the connection (*IP_UC2*), especially to differentiate traffic issued from inside the network from unexpected traffic issued from the outside which is usually blocked to prevent attacks. The third point is to filter traffic from known malicious IP networks which are blacklisted in order to avoid malware infection (*IP_UC3*). The network administrator can prevent communications to different blocks of IP address. Finally, a firewall can also filter unusual aggressive traffic toward the network both for quality of service and security purposes (*IP_UC4*). Typically, in order to mitigate a denial of service, a firewall can filter traffic from a host when the number of packets emitted per second exceeds a given threshold.

When considering the CCN paradigm, some of the aforementioned use cases derived from an IP firewall do not make sense and another must be adapted. In particular, the notion of connection is deprecated. CCN only offers a pull-based connectivity; Data can only follow the path previously set by the Interests. Any pushed traffic will be discarded by the CCN stack and does not need specific rules on the firewall. Some other use cases are still relevant but must be adapted. Indeed, filtering the traffic based on the protocol is mainly achieved based on the service port number which is not relevant in CCN. Instead, new alternative ways of filtering services based on content name should be defined. Also, filtering malicious network based on IP address blocks do not make sense in CCN. Alternatively, Data can be discarded according to the identity of their content provider.

3.3.3.2.3 CCN specific use cases

Beyond the adaptation of the usual IP firewall use cases to the CCN paradigm, a CCN firewall should also directly rely on CCN concepts to enforce the new security model proposed by CCN. In fact, a CCN firewall can filter content based on two new parameters:

- Content provider: Each content should be authenticated with its provider's signature,
- Content name: Key parameter to route content and is always given as a plain text string which should be intelligible.

As defined in [56], every CCN Data must be signed by the content provider. A CCN firewall shall check the status of the content provider and filter those known as untrustworthy or banned according to the network policy (*CCN_UC1*). Alternatively, some content may claim it is coming from a content provider however, the content is not signed by it and should also be dismissed (*CCN_UC2*). Beyond security considerations, some well-known content providers whose content is not relevant within the company network can also be filtered. This is seen for example in a website providing video-streaming which can be blocked in the same way that malicious ones are blocked in *CCN_UC1*.

The second interesting information about CCN is the fact that content name is mandatory and it makes sense. This key field shall allow the firewall to perform newly efficient filtering based on the content name (*CCN_UC3*). For instance, specific media type can be excluded from the network (.avi, .mp3, etc.). Some content whose name includes specific keywords can also be filtered. This approach can even be enhanced by a semantic analysis of the name. Both filtering (based on the content provider and on the content name) can be mixed (*CCN_UC4*) to create more fine-grained rules which can, for example, only allow a node to download executable content from few specific (trustworthy) providers. When filtering content based on the content name, the direction of the traffic is important. To avoid the leak of intellectual property by a users' mistake, the firewall can prevent some content whose type is for instance a document (.doc, .pdf, etc.), to be shared externally, while local transfers are allowed (*CCN_UC5*).

Finally, CCN nodes must also be protected against aggressive traffic rate to preserve the QoS (*CCN_UC6*). This can be achieved based on the maintained statistics of faces.

A CCN firewall can also change the default routing and caching policies of a CCN node. Based on the content name or content provider, a CCN node can decide to overcome the general rules defined by the CCN stack to only store (*CCN_UC7*) specific contents.

3.3.3.2.4 Firewall features

Based on the aforementioned use cases and of the CCN design, a first set of features for a CCN firewall is proposed in this section. By comparison to iptables, the proposed features only concern the filtering capability of the firewall. Contrary to the IP world (iptables), we do not see significant use cases that could be related to features linked to the NAT table (obviously related to IP routing) or the mangle table (overwriting fields in headers).

Previously, the core of a CCN node was presented which is composed of three tables that all have a specific purpose by opposition to the single IP forwarding table. Each one of the CCN inner tables, namely Content Store, PIT and FIB, may be restricted with a set of specific rules. Most of the following features concern the Pending Interest Table which is the key table through which every CCN packet must go through. Initially, only the matching between *Interest* and *Data* is done but the firewall introduces a lot of new features at this point. The Forward Interest Base table that stores the route to contents will not be affected by the firewall in the same way that iptables do not change directly ip routes.

Table 23 summarizes the next use cases, compares the IP and CCN world and presents the corresponding firewall features and the corresponding syntactic elements.

IP_UC	CCN_UC	Feature name	Syntactic element	Rule example
IP_UC1	CCN_UC3	Filtering on content name	*content_name*	*interest * \@**game**| **play**|**fun**\@ **15** pit drop*
IP_UC2	-	-	-	-
IP_UC3	CCN_UC1	Filtering on content providers	*provider_sign*	*data * * 0 **123456789ABCDEF;FFFF0000AAAA** pit drop*
IP_UC4	CCN_UC6	Filtering on heavy traffic	*r_face*	***face 200***
-	CCN_UC2	Filtering on bad signature	*sign_chek*	*data * * **1** * pit drop*
-	CCN_UC4	Composition of filters	*match_data*	*data * \@**game**| **fun**\ @ **0 0** **123456789ABCDEF;FFFF0000AAAA** pit drop*
-	CCN_UC5	Filtering on content direction	*direction*	*interest **in** \@\ .doc$\@ 0 pit drop*
-	CCN_UC67	Filtering of stored data	*"cs"*	*data * * 1 * **cs** drop*

Table 23: Relationship between use cases, firewall features and language

(The bold part in the rule example corresponds to the main syntactic element involved in the use case)

Filtering on content providers: the signature of the content provider is mandatory for each CCN Data packet. More precisely, every content creates a mapping triple: $M_{(Name,Cotent,Provider)}$=(Name,Content,Sign $_{P(N,C)}$). The signature includes a cryptographic digest encrypted by the publisher's private key, the public key itself or its location. To filter traffic based on the content provider, the firewall must retrieve information on the provider's key from the signature and compare it with a blacklist of keys. Alternatively, to improve the security, a whitelist can also be created so that only content from authorized providers will pass through. Data packets that fail the check will not be transmitted to the face and the corresponding Pending Interest is deleted.

Filtering on bad signature: The validity of the signature can also be checked while processing the signature. When the firewall is given the provider key, the content and its name and is not able to find the same fingerprint, the content can be discarded as its source cannot be checked.

Filtering on content name: As illustrated in section 2.3.1, the two types of CCN packets includes the name of the content as a plain-text information. By analyzing the different keywords composing the name, the firewall thanks to regular expressions can filter content based on a blacklist of the different pieces of information composing the content name, more precisely: the name prefix, keywords and the file extension. Interest packets that fail the check will not be written in the Pending Interest Table and Data packets that fail will not be transmitted to the requesting face. The corresponding Pending Interest is deleted.

It would be useful to filter all content with a given semantic, for example, to prohibit paedophilia-related content. Thus, our firewall can consider additional words that are semantically close to those defined by the user in the firewall rules. An allowed semantic distance can also be defined by the user.

Composition of filters: The two previous parameters (content provider and content name) must be considered together to allow the firewall to use more complex rules that can filter, or authorize, specific content type or content name according to the provider.

Filtering on content direction: The firewall should be able to differentiate local traffic (*Data* published locally or under the company's prefix) from external traffic.

Filtering on heavy traffic: The CCN stack includes a strategy layer which is in charge of maintaining statistics for each active communication faces in order to select the best available communication channel. Based on these available statistics, the firewall can detect faces showing an abnormal activity in case of bottleneck and prevents DoS. This can be achieved by decreasing the rate of Interests written in the PIT by aggressive communications. Alternatively, as Interests are also used to control traffic stream, like a reception window, the firewall could overwrite the field of cumulative requested Interests to reduce the consumed bandwidth.

Filtering of stored data: The caching policy can easily be enforced by the firewall. To proceed, the same rules written with the firewall language can be applied to the Content Store in addition to the PIT. More precisely, if a CCN Data packet is eligible to be stored in the CS according to the caching algorithm (most recent, most frequent, etc.), the firewall can then check the provider signature and/or the content name according to the defined rules before caching the Data.

3.3.3.2.5 Rules definition language

For the ease of use and the readability of the rules, our syntax has been taken and inspired from the one used in iptables.

Hence, the firewall rules can be defined according to the following grammar expressed using the Augmented Backus Naur Form [62].

```
rule = r_interest | r_data | r_face
r_interest = "interest" SP direction SP match_interest SP "pit" SP action
r_data = "data" SP direction SP match_data SP ["cs"|"pit"] SP action
r_face = "face" SP number
direction = "*"|"int"|"ext"
action = "forward"|"drop"
match_interest = content_name
match_data = content_name SP provider
content_name = "*"|reg_exp
provider = sign_check SP provider_sign
sign_check = "0" | "1"
provider_sign = "*"|first_sign *next_signs
first_sign = hex_value
next_signs = ";" hex_value
reg_exp = "\@" re_posix "\@" number
re_posix = <a standard posix regexp>
hex_value = 1*hex
SP = 1*%d32 ; one or more space characters
hex = "A" | "B" | "C" | "D" | "E" | "F"| "a" | "b" | "c" | "d" | "e" | "f" | digit
number = 1*digit
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

As highlighted, there are three possible rules: for filtering interests (r_interest), data (r_data) or faces (r_face). In fact, the last case allows the firewall to filter heavy traffic by limiting the global number of packets per second defined by “number”. For other types of rules, we can first filter on the direction as mentioned in the previous section using “int" or "ext" to indicate respectively whether the content is or was asked by the node itself or another one. Also, the actions are the same for these rules, *i.e.* "forward" or "drop" on the PIT table which means that the firewall will forward or not forward specific Interests which is equivalent to add or not add an Interest in the PIT table. The firewall can decide to drop certain Data packets. At a first glance, it might be unfair to drop such Data since it means that the corresponding interest has been accepted previously. However, an Interest can return Data with a different name when using the dynamic generation of content upon request, as mentioned in section 2.3. In addition, a rule on Data can also control the caching policy by preceding the action by “cs". An Interest can only be matched regarding the requested content. It is basically a standard POSIX regular expression delimited by "@" where “number” defines the level to consider for extending semantically the regular expression.

For filtering content, it can also be done regarding the provider by checking if the signature is valid ({sign_check) and/or on the provider itself identified by its signature, {provider_sign which is a sequence of hexadecimal characters. As noticed in the grammar, the space characters (SP) is used. There are also several rules using the character * which is a shortcut

for a regular expression matching everything and avoiding the usage of the delimiter "@".

In Table 1, some examples show how to construct rules related to the different use cases. For example for *CCN_UC1*, data signed by the keys *123456789ABCDEF* and *FFFF0000AAAA* are blacklisted. In order to understand the concept of direction, the example in the table means that the node where the firewall is instantiated cannot share files with the extension *.doc* as the firewall will drop corresponding interest when they are associated to internal content “in”. However, the node can forward interests about doc files of other nodes since there is no restriction using “ext”.

3.3.3.3 Firewall architecture

To implement the features defined in the language, the firewall provides tools, modifies the CCNx library and relies on a deployment strategy described in this section.

3.3.3.3.1 Semantic preprocessing

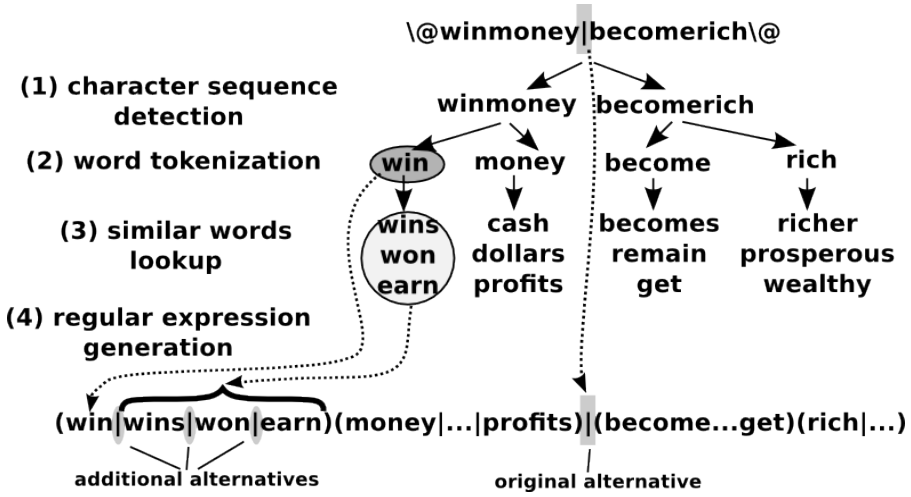

Figure 25: Semantic extensions -- short example

As highlighted in previous sections, filtering on content name uses regular expressions which can also contain meaningful words like money, game, humour, etc. Assuming the goal is to filter names like *winmoney* or *becomerich* which, in the current Internet, could be used in a URL to attract people to malicious websites hosting malware, the semantic extension corresponds to also filter related content names like for example: *wineuros*, *windollars*, *earnmoney*, *bericher*, etc. This is essential in CCN as it provides a great freedom in naming and announcing content. To accomplish that, there are four steps as highlighted in figure 22:

1. Sequences of 3 or more characters are extracted. This allows to only extract potential meaning full words by discarding digits or symbols of the regular expression like {, } or ^ as well as irrelevant very short sequences of characters.
2. Sequences of characters are segmented as real human readable words
3. For each extracted word, a list of similar ones is created
4. For each list of similar word, the regular expression with alternatives is created integrating the original word as well and replaces the original word itself in the original regular expression. This is illustrated in figure 22 where the original pipe character (surrounded by a gray square) is kept.

For the word segmentation, the technique proposed in [63] is employed. It finds the optimal

word segmentation by successively dividing the sequence of characters in two parts while multiplying the probability of each word corresponding to their frequency in text samples.

Searching for a similar word is done through DISCO [64] relying also on text samples like Wikipedia. It computes the relatedness between two words *w1* and *w2* by finding the number of co-occurrence of these words with a third one *w3*. Assuming all possible *w3* which are close to *w1* and *w2* (2 intermediate words maximum), the relatedness is computed using the mutual information metric. Using the same method, DISCO is able to get the *n* most similar words to another.

Therefore, when a strictly positive number *n* is specified after a regular expression this leads to generate the n most similar words of each word extracted at step 2 in the previously described process. It is important to note that some irrelevant word might not have been filtered by the first step like ``abcde'' but in this case DISCO will not find any similar word and so alternatives will be generated in the final regular expression.

As such semantic algorithms are resources consuming, they cannot be executed when the firewall is running. Therefore, it has to be done as a pre-processing, *i.e.* the original rules are read and extended.

3.3.3.3.2 Integration within the CCNx library

Place of the firewall within the CCN stack: Given the CCN stack scheme from [56] our firewall works within the Security layer and processes directly content chunks as illustrated in Fig. 23. The Security layer as defined in [43] will handle authenticity of the incoming chunks and our firewall can request signature verification which it will handle and then give the result to our implementation.

In order to provide some flexibility and modularity, the code of our firewall has been integrated within the CCN demon so that it is triggered when the CCN inner tables (PIT and CS) should be modified. This induces only minor changes to the CCN standard implementation as well as provides flexibility for further updates. Our firewall captures a packet once it arrives from a face and then applies any given rule before letting the standard process apply performing updates on the tables.

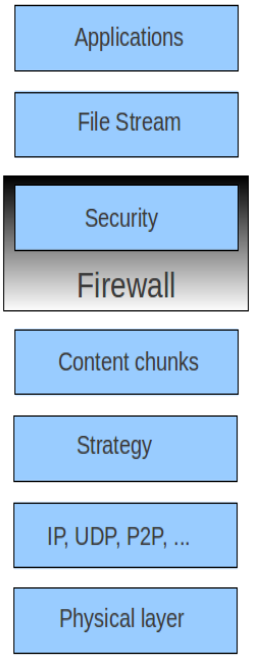

Figure 26: Firewall implementation within CCN stack

Implementation: In order to integrate the rules preprocessed by DISCO, a simple linked list which holds the rules in the memory has been created. These are loaded depending on what type of rule they correspond to i.e. a face rule will be stored differently or an interest rule. Furthermore, there is a matching algorithm which checks if one or more rules matches with a given CCN Data or Interest packet. These functions are provided in the *ccnd_firewall.c* and *ccnd_firewall.h* files.

In order to load the rules at runtime the *ccnd.c* source file was modified such that when a new instance of ccnd is created, by the method *ccnd_create*, it will load the rules stored in a text file whose path is given as parameter. The *ccndstart* script launching the demon was also modified such that it allowed us to indicate the path to the rules.

A piece of code in *ccnd.c* was finally integrated which is called each time a CCN packet is processed. For our matching algorithm call in the *process_incoming_interest* was included and it will check if the interest just being processed matched any rule provided beforehand. The same method was applied but focused on data packets in the *process_incoming_data* method.

3.3.3.3.3 Management

Default security policy: One of the first things to do when configuring an IP firewall is to set the default security policy that will be applied in the absence of matching rules. The settings range from the most restrictive rule: *deny all, accept on match*, to the most permissive rule: *accept all, deny on match*. For our CCN firewall, we also provide the administrator with such configurations. The recommended default security policy is to deny all, except packets signed with the public key of the network administrator. This single exception is to give a way for a node to always have its firewall rules up to date.

Deployment and administration: To tackle security issues created because of nodes mobility in companies (laptop computers, smart-phones, etc.), we recommend a deployment of the firewall on each node rather than only on nodes providing the Internet connectivity. Upon installation, the firewall software is given the public key of the network administrator to be able to authenticate its content. Then, the rules implementing the security policies of the company can be updated on a regular basis. Firewall rules are defined as a content that is

requested by the firewall periodically. The rules are applied if properly signed by the network administrator.

3.3.3.4 Experiments

3.3.3.4.1 Performance evaluation

To test our implementation we used the following setup as illustrated in Fig. 24. The architecture consisted of six CCN nodes arranged and configured such that the packets need to go through all nodes to be delivered. Each node is a standard workstation with an Intel Pentium 4 CPU running at 3 GHz and 2 GB of RAM. For the evaluation, ccnx 0.6.0 release was used as reference.

One node named "Content provider" provides the content used for our evaluation, some simple raw files. The intermediate nodes served merely as routers without caching repository. Their function was only to allow the "Consumer" node to reach the content provider. Finally the last node on the path was the node requesting our testing content. This is also the measuring point for our evaluation. Our firewall implementation was deployed on all the nodes, to follow the recommended deployment strategy. Two evaluations based on this setup were performed, each time starting CCN on all nodes with the same route configuration and requesting the same content.

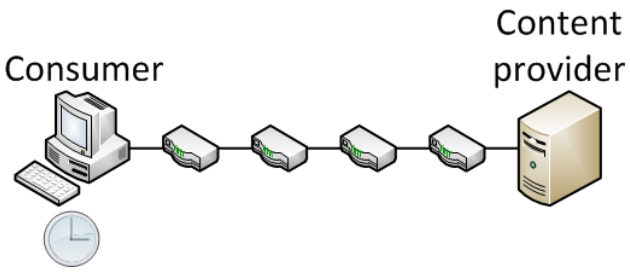

Figure 27: CCN firewall evaluation architecture

In a first attempt to evaluate the performance of our implementation, the end-to-end transfer time for two different binary files with a size of 500 MB and 1 GB were measured. The files were requested several times but for each time the number of rules were increased to see if there is an impact on the transmission time. The rules are designed so that the Data packet is accepted when matching the very last rule so that we are certain that they are all processed. The results obtained from this evaluation are given in Fig. 25. It shows that there is no visible impact when our firewall is applied onto the standard CCN implementation, even on the most demanding configuration (1000 rules on each node, 1GB file).

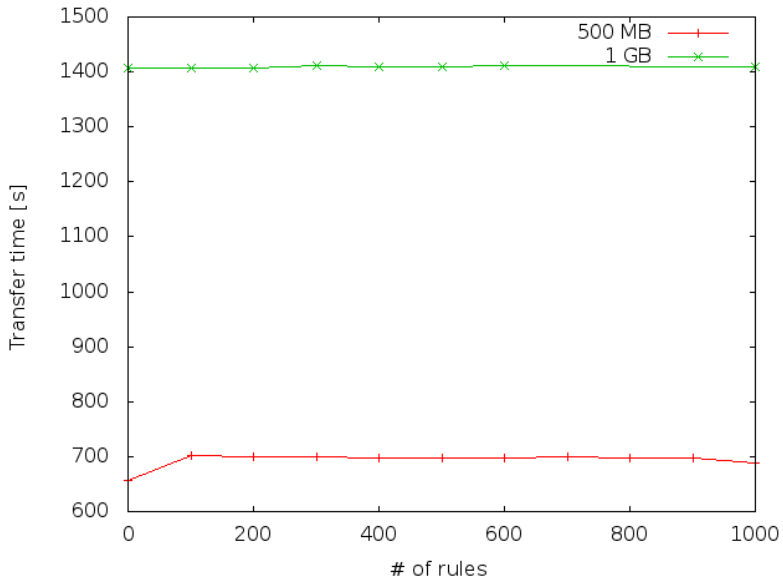

Figure 28: Impact of the number of rules on the transfer time

A second evaluation was performed in order to compare the performance of a standard CCN implementation and another including our firewall. This experiment was performed several times to obtain significant values for a later comparison with our implementation. For each observation, the same binary file of 500 MB was transferred in order to obtain a mean transfer time. Later on, several experiments were performed, each with a fixed number of 1000 rules, with our implementation in order to have comparative values to the standard CCN implementation.

The results are shown in Fig. 26. The end-to-end transfer time was measured from the content provider to the consumer node. The obtained values were checked if they represented expected results by applying a normal distribution. For this purpose, the mean transfer time and the standard deviation for both setups were calculated. The results are visible in Table 2.

	Standard CCN	Firewall modification
mean	684.9718	697.1672
std. deviation	1.3463	1.5437

Table 24: Observation results

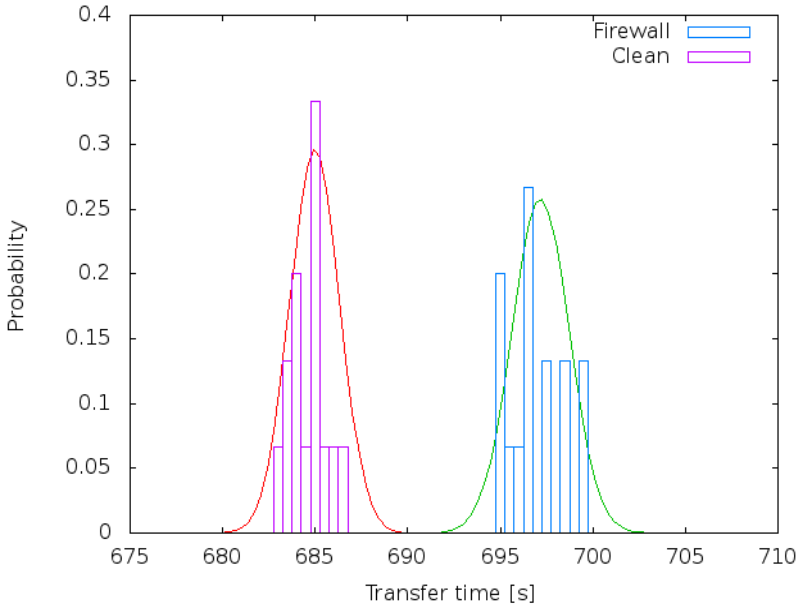

Figure 29: Impact of the firewall with 1000 rules on the transfer time

As shown from the above graph, both the clean CCN implementation and our additional modification to it follow a normal distribution and was confirmed by applying Chi-square and KS-test on both measured sets obtaining a confidence level of 91% for our implementation and 85% for the standard CCN implementation. From our measured observation, it can be deduced that the average overhead on the transfer time induced by our implementation is the difference between both averages: 697.17 - 684.97 = 12.20 s, which corresponds to $1.75%.

3.3.3.5 Conclusion

In this section, another way to improve the security of CCN is proposed by designing a first dedicated firewall, from the use case study of the security needs of a CCN administrator, to the grammar definition and the implementation within CCNx. Since this paradigm relies on content based routing, where the name of the content and its authentication are key aspects, these opportunities were used to provides innovative features compared to a regular IP firewall. In particular, to leverage semantic tools was proposed. Performances evaluation is very promising since the message throughput degradation is lower than 2%. In future work, we plan to investigate potential optimization, for example by being inspired by rule reordering research for standard firewalls or using bloom filters for accelerating rule lookups. The last section on CCN security will deal with the key management in such networks.

3.3.4 CCN distributed key management

3.3.4.1 Introduction

An important challenge regarding the security of CCN is the authentication of content through the management of encryption keys linked to content providers, as described in the security model proposed for CCN [43]. In fact, to secure content diffusion despite the fact that content can be retrieved from any node, CCN proposes a new security model based on the authentication on every piece of content, heavily relying on encryption. If some directions have already been proposed in some reference papers from PARC [43,47], the key management scheme to implement such a security strategy is yet to be designed. However, such a key management scheme that can support every communicating device is a real challenge by the scale of the needed deployment. The nodes should be able to generate the

key material to certify their Data and retrieve the key material from others to authenticate the received Data. To be scalable, the system should also provide a way to authenticate the encryption keys as well.

Thus, to keep the privacy and integrity of content using encryption, a proper key management scheme is necessary which authenticates the encryption key as well. In this section, an online public key generation technique which is suited for the CCN architecture is proposed. We validate the proposed technique using the ccnSim simulator to evaluate its performance and the AVISPA tool to assess its security.

3.3.4.2 Related Work on Key Management Schemes

Since there has been little investigation made in securing CCN as a result there is no specific key management scheme for it. Therefore, some of the existing key management schemes proposed in the literature for the static and mobile ad hoc networks which is a field of networking where key dissemination in the network has been investigated for years are described.

Basically, cryptography is divided into two main categories (1) Symmetric key cryptography and (2) Asymmetric key cryptography. In symmetric key cryptography, every node in the network is assigned *N-1* keys where *N* is the total number of nodes in the networks. This is not suitable for a large network because each node is required to store large number of keys. In asymmetric cryptography, each node is assigned a pair of keys (i.e. public key and private key) for secure communication with other nodes in the network. Recent research works in cryptography are mainly based on the traditional public key infrastructure (PKI: [51],[52],[53],[54]), and identity based public key cryptography (ID-PKG:[55],[56]). In the following section, the compatibility of those approaches with CCNs is discussed.

Smetters in [47] suggested the standard PKI approach for CCNs. In this approach, each node has a pair of keys (public key and private key) and for secure communication, each node publishes its public key along with its certificate, assigned by a certification authority. Each node in the network verifies the public key of other nodes by sending the attached certificate to the certification authority for validation. The certification authority checks the validity of the certificate and sends the acknowledgement back to the node about the authenticity of the certificate. Since PKI is based on the concept of a single centralized certification authority, it does not suit well the concept of CCNs, where contents are replicated in the network and are requested and routed by name instead of being provided by a single source. This increases the number of public/private key pairs, which in turn increases the verification overhead for the single certification authority.

ID-PKG completely eliminates the need for public key certificates by exploiting publicly known user identity information (such as IP address or telephone number) as a public key for securing information. ID-PKG enables any pair of users to communicate securely without exchanging public key certificates, and without using the online services of a third party. This is enabled by a trusted Private Key Generator (PKG), which generates the private keys of the entities using their public keys and a master secret key. In 2003, the first ID-PKG cryptography management and certification scheme [55] for mobile ad hoc networks was presented by Khalili and Katz. The basic idea of the scheme is similar to the scheme of Zhou and Haas [51]. A group of selected nodes shares the responsibility of managing the PKG. Each node can obtain a system private key share from a predefined minimum number of PKG manager nodes, which collaborate with the node to generate the private key.

This scheme has several main problems. (1) These papers did not mention many potential problems that can arise on the channel between PKG nodes and user nodes. Consider for example the fact that a user's private key could be easily wire tapped by attackers. (2) The scheme does not mention how to identify the nodes that manage the PKG and how to gain

their private key when new nodes are added to the network. (3) The scheme does not explain how to update the main key pair of the system (including public and private key).

Hence in 2004, Deng proposed a new cryptography management and certification scheme called ID-PKC [56]. In this scheme, there is a master public/private key pair. The master public key is known by all nodes in the network, while the master private key is divided into shares and distributed among *k* nodes of the network (fewer than the total number of nodes). Each node ID is working as node public key and for secure communication, it needs its private key. So the node sends a request to *k* PKGs to get its private key share. The requesting node generates a temporary public/private key pair and gives that public key to the PKGs to get its encrypted private key share. After getting those shares, the node generates its private key. Then the PKGs announce the requesting node ID to all nodes of the network, to be used as its public key. This scheme, however, still does not address the problem of updating the main key of the system.

The system based on ID-PKC is a powerful alternative to PKI in terms of both efficiency and convenience. However, it also has several problems, such as the key escrow problem in case of PKGs compromission and suffers from a single point of failure (as shown in [56]), because there is only one PKG responsible for generating the private keys to the nodes. Although this second problem was solved by [55], the key escrow problem still exists, because the private key of the PKG is compromised, the entire system is compromised.

3.3.4.3 Architecture of the Key Management Scheme

The existing key management schemes for TCP/IP networks secure the links from source nodes to the destination nodes irrespective of the number and type of packets/data. Hence these schemes are ill suited for the CCNs architecture, where there is no concept of link between the requesting node and content generating node.

3.3.4.3.1 Design principles

In the standard PKI approach, there is a certification authority and when the destination node receives the public key of the source node, it validates the received key using the certification authority. But this scheme is not suitable for content centric networks, where the keys are related directly to the contents, instead of the source ID or location. Also in the standard PKI approach, a node can use its encryption key (private key) to encrypt all the content and the destination node needs to verify the decryption key (public key) with the certification authority to check the authenticity and integrity of all the received contents. But in content centric networks, there is no concept of content source ID/location information, hence each received content key should be verified with the certification authority, which increases the overhead on the network and the time required to verify the key. Hence in this deliverable we propose to use the ideas of using: (1) distributed key holding nodes to reduce the communication overhead on a single node and (2) key shares to check the authenticity and integrity of the decryption key as well as of the received content.

In fact, since cryptography is considered as the main building block of any security primitive, the cryptographic keys should also be secured and authentic. To this aim, the key management scheme should be secure and each node of the network should be able to authenticate the cryptographic key(s). This is the most challenging problem in CCNs, where the keys are linked with the content names instead of the content generation source. Hence, we have tried to solve the problem in our proposed key management scheme for the CCN networks which is not possible by the existing key management schemes for the traditional TCP/IP networks.

Thus, an authentication and key establishment scheme for CCNs is proposed in which the contents are authenticated by the content generating node, using pre-distributed shares of encryption keys. The content requesting node can get those shares from any node in the

network, even from untrustworthy ones, in accordance with a key concept of CCNs. In our work, means to protect the distributed shares from modification by these malicious/intruder node has also been provided. In the following section, the assumptions on the architecture of Content Centric Networking are described, and then what follows is a description in detail of the proposed key management scheme.

3.3.4.3.2 Network Architecture

Since the Internet is a composition of a large number of small networks as shown in Figure 27, the assumption is that each individual network has its own network manager, which are powerful secure nodes which act like servers for management and security-related aspects of networking.

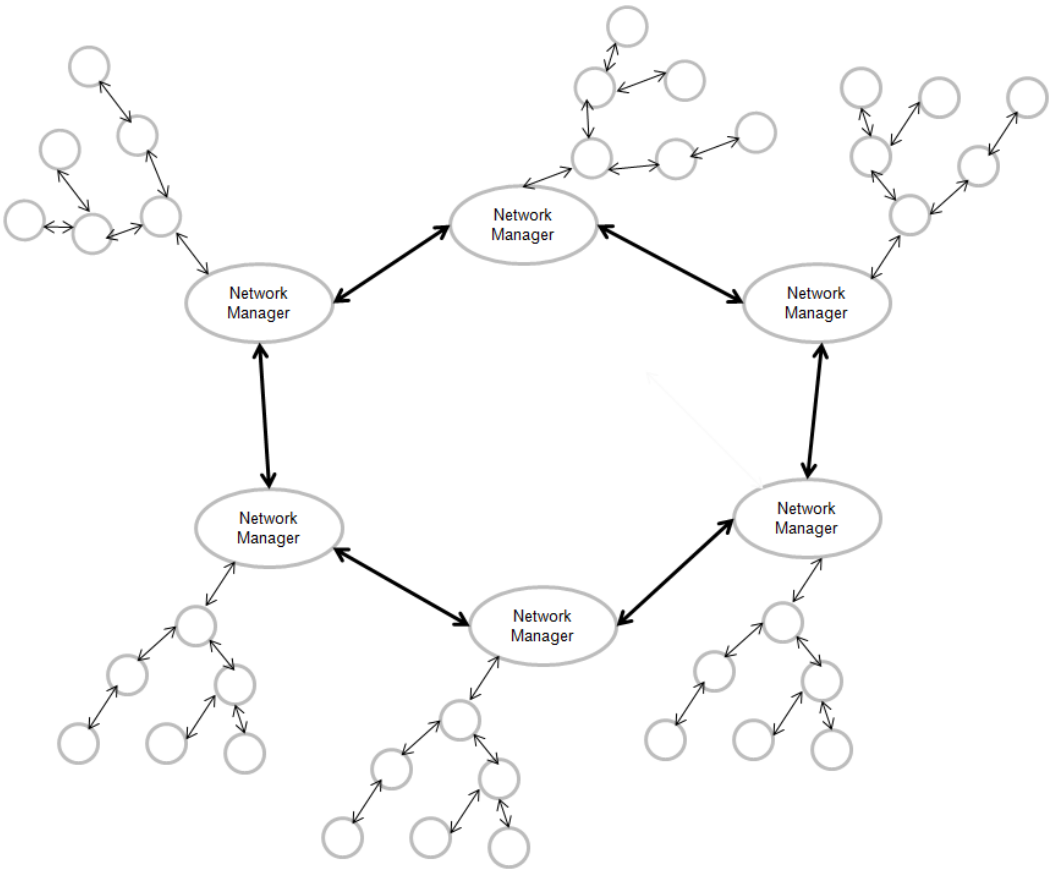

Figure 30: Virtual internet architecture

Each small network consists in a large number of nodes. All the nodes of each small network are divided into two different categories, i.e. (1) Normal Nodes (NN) and (2) Key Holding Nodes (KHN). Both types of nodes are similar in terms of capabilities and architecture. The KHNs are responsible for initially holding the key materials of the encryption key(s) related to the content(s) once they are generated by any node (source) of the network after the network deployment.

The selection of KHNs is based on the maximum number of connections established by a node with its neighbouring nodes. This approach minimizes the initial network traffic due to key management, since each node will be at a maximum of two hops from a KHN. Note that when nodes are deployed they are all assigned the same security-related material and hence can play both roles. The distinction between KHN and NN is made after deployment, when the node joins a network, and only affects the role that the node plays in providing and using the key material related to the content diffused.

In order to select KHNs, each node shares its connections count with its neighbouring nodes. Once all the neighbouring nodes receive those count, each node become aware of its neighbouring node's and considers it with the highest or equal connections as its nearer KHN.

Figure 28 shows the virtual organization of KHN and NN in the network. It should be clear from the figure, that the nodes 3, 4 and 8 have the highest number of connections with their neighbouring nodes, so they act as actual KHNs. On the other hand, nodes 2, 5, 7, 9 are not selected as KHNs, but can still act in this role for NNs that are two or more hops away from the actual KHNs, and so on.

Each node (both KHN and NN, since also the former can generate contents) is also assigned some key materials to generate their public/private key pair for securing the generated content. The assignment of those key materials to the nodes is performed off-line while the assignment of key materials related to the content(s) to the KHNs is performed on-line. The network manager also plays an important role in generating the key materials for its network nodes.

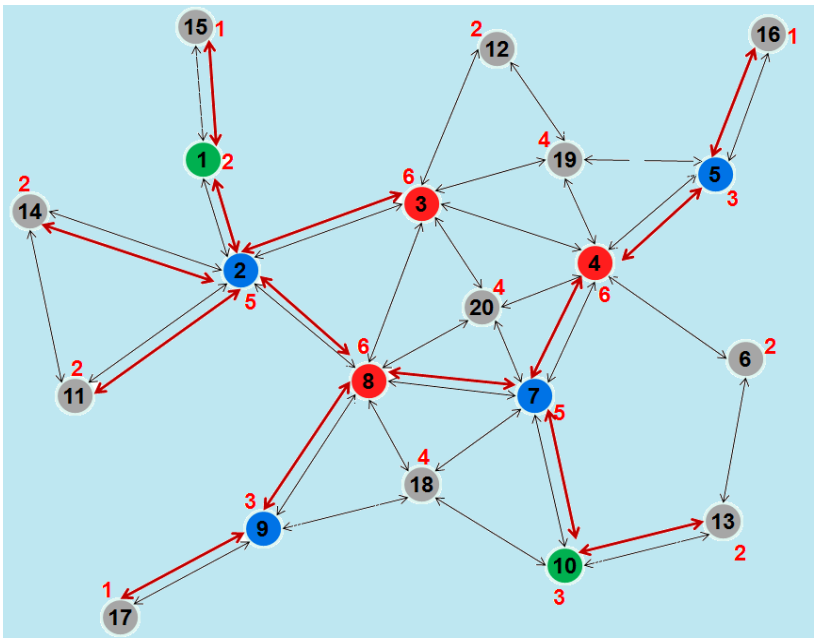

Figure 31: Virtual organization of Key Holding Nodes and Normal Node in the network

3.3.4.3.3 Key Material Assignment

Each node in the network is assigned some important key materials which are used in generating the public/private key pair in order to secure the generated contents. More specifically,

- Each node in the network is assigned a random number generator, a one way Hash function (H), a share generation function (f) and a natural number group generator G (G can be, for example, a prime number).
- Each network manager is also assigned a fixed random number (NMRN) assigned by the network owner, a random number generator, a one way Hash function (H), a share generation function (f) and a group generator G.

3.3.4.3.4 Key Establishment and Management

Once the network is deployed, each node in the network sends a join message to its network

manager. After the reception of those "join" messages, the network manager sends a fix random number (NDRN) and a NeTwork Public Share (NTPS) to each joining node. The network public share is generated as

$$NTPS = f(NMRN, node\ ID, RN) \quad (1)$$

Where RN is a random number computed from the generator. When a node receives the random number and NTPS from its network manager, it generate a NoDe Public Share (NDPS) of its public key as

$$NDPS = f(NDRN, RN, NTPS) \quad (2)$$

In CCNs, since contents are requested by their names instead of their generating source, there must be a relationship between the content and its encryption/validation key. Hence, we introduce two further shares generated by the source node of the content in order to relate the encryption/validation key with the content. Those two shares are P_1 and P_2 which act as the two parts of the public key for a content. These two shares (P_1,P_2) are generated as

$$P_1 = f(NTPS + Content) \quad (3)$$

$$P_2 = f(NDPS + P_1) \quad (4)$$

The required public key k_{plc} is

$$k_{plc} = P_1 + P_2 \quad (5)$$

Since each node is given a group with a generator G, it selects a random number g from G and also creates the hash of the content-related public key shares and the corresponding private key $ k_{prt}$ as

$$X = H(P_1),\ Y = H(P_2),\ Z = H(k_{plc}) \quad (6)$$

$$k_{prt} = K_{plc}^{-1}\ mod\ G \quad (7)$$

The node also calculates *A*,*B* and *C* for the authentication of the generated content and its shares as

$$A = g^X,\ B = g^Y,\ C = g^Z \quad (8)$$

After the generation of the public key shares and their hashes, the node distributes those shares among the nodes (KHNs) responsible for holding those shares i.e. (*P_1, P_2, C, Z, NDPS*). The KHNs get the NTPS of the received NSPS from the network manager. The node includes *A* and *B* in the data packet along with the hash of content in order to help the destination node get and verify the public key shares i.e. (*Content, A, B, Z*).

If now a node receives a data packet containing the content and hashes for the authentication, it needs the public key shares to verify those received hashes. In order to get the public key share from the KHNs, it sends a key share request to the KHN nodes. The KHN sends (*NDPS, NTPS,C*) to the requesting node. After receiving this message, the node generates P_1 and P_2 using (3) and (4) and the share generation function *f*. After the generation of P_1 and P_2, the node generates *X* and *Y* using (6). Now the node calculates (C^X, C^Y) using the received *C* and calculated *X* and *Y* and then compares them with the (A^Z,B^Z) received in the data packet. Successful verification authenticates the received messages and the contained public key shares.

3.3.4.4 Performance analysis

The performance of our proposed scheme for content centric networks in terms of time taken by a node to retrieve a key is evaluated. It is compared with the standard PKI approach for key establishment and management. To this aim, the ccnSim simulator [57] developed specifically for content centric networks is used.

3.3.4.4.1 Simulation scenarios

In order to evaluate the performance of the proposed scheme against the standard PKI approach, different network topologies provided in the ccnSim simulator is used and the average time taken by a node to retrieve the key(s) for the received content(s) is noted. To do so, each node in the network generates a content which is composed of 100 files. Each file is encrypted by a separate key and the corresponding key shares are distributed among the Key Holding Nodes (KHNs). Also each file is split into five chunks. When a node starts receiving the requested file after sending an interest for that file, it waits until all the chunks of the requested file arrive. Once the requested file is completely received, the node (requester) sends a request for the key shares of the received file. After the reception of those key shares from the nearest KHN (all others will be discarded, according to the CCN principle), the requester verifies the authenticity and integrity of the received file.

3.3.4.4.2 Results

During the simulation, one node is selected as a key holding node for the standard PKI approach and three nodes as key holding nodes for the proposed scheme. Table 3 shows the average time taken by a node in different network topologies to retrieve a key for the received content. Our scheme is at least as good as a central authority and up to 50% faster to retrieve the key in some specific topologies.

Scheme	Geant topology (s)	Level 3 topology (s)	Tiger topology (s)	Dtelecom topo (s)
PKI	0.009	0.020	0.0003	0.0133
Our	0.004	0.018	0.0002	0.0131

Table 25: Average time taken by a node to retrieve a key for content in different network topologies

We compared the *Hit rate* of the proposed scheme with that of the standard PKI approach. The *Hit rate* is the ratio between (1) the number of key interests received by a node for which the node has keys stored in its memory and (2) the total number of key interests received by that node.

$$Hit\ rate = Hit / (Hit+Miss)$$

Where *Hit* counts how many times a node has a key for the received key interest packet and *Miss*counts how many times a node does not have a key for the received key interest. The ccnSim simulation results in Table 4 show that our solution has a better hit rate than the standard PKI, improving the keys disponibility in the network.

Topology	PKI Approach	Our Approach
Geant	0.92	0.93
Level3	0.88	0.91
Tiger	0.96	0.97
dtelecom	0.93	0.94

Table 26: Average Hitrate of each nodes in different network topologies

3.3.4.5 Security analysis

Since a relationship was kept between the content and its encryption key using the NTPS and NDPS, only the authentic nodes of the network are able to encrypt the content(s) using the authentic key(s). An adversary would not be able to encrypt the content using the authentic key of the network until and unless it becomes an authentic member of the network (i.e. it receives the key material described above before deployment). On the other hand, by using the standard PKI approach for CCNs, an adversary can generate and encrypt fake content using its generated private key corresponding to an authentic public key. This is easy because in CCNs key request is based on the content name instead of the generating source.

Also we eliminated the need for a centralize certification authority for the verification of the encryption keys by using the key share concept and by securing the keys with the network and node parameters. So the adversary would not be able to generate a fake private key simply thanks to the non-existence of a single complete public key.

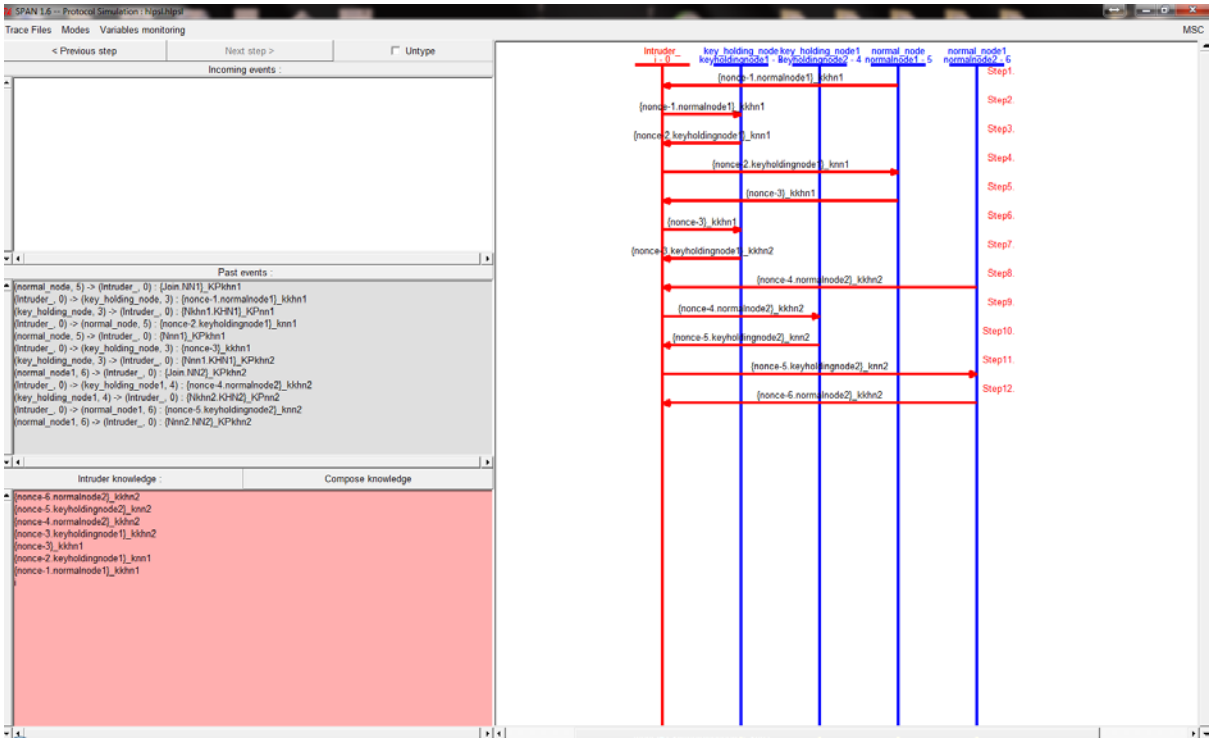

Figure *32*: *AVISPA tool screenshot*

In order to validate the secrecy of the proposed key management scheme for content centric networks, the AVISPA (Automated Validation of Internet Security Protocols and Applications) tool was used. AVISPA is a push-button tool for the automated validation of Internet security-sensitive protocols and applications. It provides a modular and expressive formal language for specifying protocols and their security properties, and integrates different back-ends that implement a variety of state-of-the-art automatic analysis techniques (e.g.

OFMC, ATSE, etc).

We implemented the proposed key management scheme in AVISPA and checked its security using some of the attacks provided by AVISPA, namely OFMC (On-the-Fly Model-Checker) and CL-AtSe (Constraint-Logic-based Attack Searcher). The former builds the infinite tree defined by the protocol analysis problem in a demand-driven way, i.e. on-the-fly and uses a number of symbolic techniques to represent the state-space. The latter provides a translation from any security protocol specification written as transition relation into a set of constraints which can be effectively used to find attacks on protocols. Both translation and checking are fully automatic and internally performed by CL-AtSe, i.e. no external tool is used. In this approach, each protocol step is modelled by constraints on the adversary knowledge. These results are shown in Table 3 and tend to prove that our solution does not lack from security issues.

Technique	Summary
OFMC	SAFE
CL-AtSe	SAFE

Table 27: AVISPA Simulation Results

Also Figure 29 shows the interface of the AVISPA tool, illustrating a case in which the intruder acts as a man-in-the-middle. The intruder in this case acquires the key share but does not have the functions for generating the complete public key and the key shares ensuring authenticity, hence it cannot modify the actual content, whose integrity and authenticity are still ensured.

The proposed scheme hence follows the basic concept of content centric networks, which states that a node can get the required authentic keys and contents from any node of the network, even an intruder/attacker. This means that the key management scheme must be so secure that the intermediate nodes/attackers cannot modify the contents/keys which are provided by our proposed scheme. Even if a node gets the required shares from the intruder who acts as a man-in-the-middle, a node is able to verify and authenticate those shares.

3.3.4.6 Conclusion

As CCN heavily rely on content authentication to create a new web of trust, a scalable and secure key management scheme is mandatory for the further development of CCN. These two constraints were addressed by designing the proposed key management system: (1) The introduction of key holding nodes to reduce the communication overhead used for authentication and (2) key shares to check the authenticity and integrity of the decryption key as well as of the received content. Performance simulation on ccnSim showed that our scheme is at least as good as a central authority while its distributed nature makes it more scalable. Thus, the security analysis performed thanks to AVISPA tends to prove that no security leak exists.

Finally, in this section 2.3, three different studies were presented providing critical elements for improving the security of Content Centric Networking: (1) a monitoring architecture, (2) a firewall to enforce security policies and (3) a key management scheme. Our future work will consist in the deployment of a full CCN stack on Contiki-capable sensors and the evaluation of our security elements on a sensor network to show their applicability in the context of IoT communications. At the same time, a comparison will be made to this new routing layer with the 6lowpan stack when supporting pervasive services.

4 Privacy

4.1 Encryption

Encryption is a way to hide the meaning of a data message by running the message through an encryption mechanism. Encryption mechanisms, sometimes called ciphers, take the data to be encrypted (the plaintext) and an encryption key to form an encrypted form of the plaintext (the ciphertext). The ciphertext is decrypted by running it through a decryption mechanism. Like the encryption mechanism, the decryption mechanism also needs a key to be able to turn the ciphertext back into plaintext.

Encryption mechanisms can be divided into *symmetric* and *asymmetric* mechanisms. Symmetric mechanisms are considered symmetric because the same key used to encrypt the message can also be used to decrypt it. Two parties that use a symmetric cipher to ensure confidentiality of their communication thus require a shared key. There are a number of symmetric encryption algorithms available. The most common ones are the American Encryption Standard (AES) and the Digital Encryption Standard (DES).

Unlike symmetric encryption algorithms, asymmetric algorithms use different keys for encryption and encryption. Asymmetric algorithms allow two communicating parties to use non-shared keys when protecting the confidentiality of their communication. There are several asymmetric encryption algorithms. The most well-known example is the Rivest-Shamir-Adleman (RSA) algorithm. Asymmetric algorithms are used in so-called public key encryption systems. In this system, every party keeps two keys: one private and one public. The public and private keys are used as encryption and decryption keys in an asymmetric encryption algorithm. Depending on how the messages are encrypted and decrypted, the private and public keys alternate as encryption and decryption keys.

4.2 Key management issues in Wireless Sensor Networks

Before a WSN can exchange data securely, encryption keys must be established and distributed among sensor nodes.

Key management is a broader term for key distribution, which also includes the processes of key setup, the initial distribution of keys, and key revocation (i.e., the removal of compromised key). Key management is still a challenging issue in wireless sensor networks (WSNs), due to the constraints of sensor node resources[33].

Key management, like security, is a cross-layered issue. The need for key management starts at the link layer, where most of the time, the IEEE802.15.4 standard[19] will be adopted in a WSN. Although this standard considers key usage for secure data transmission, it does not specify how to exchange keys securely. Thus, this leaves open the key management problem.

Besides the link layer, upper layers such as the network and application layers also must exchange keys securely. Many security-critical applications depend on key management processes to operate but also demand a high level of *fault tolerance* when anode is compromised. This is a challenging problem because there are many stringent requirements for key management, and the resources available to implement such processes are highly constrained.

4.2.1 Security and operational requirements for Key management

Key management requirements include: (i) security requirements that are a subset of the overall WSN security requirements and (ii) operational requirements that act as constraints in the design and realization of key management.

The comprehensive security requirements of a WSN are summarized in Table 28.

Requirement	Description
Confidentiality	Nodes **should not** reveal any data to unintended recipients.
Integrity	Data **should not** be changed between transmissions due to the environment or malicious activities.
Data freshness	Old data **should not** be used as new (i.e., prevent replay attacks).
Authentication	Data used in decision-making processes **must** originate from the correct source.
Robustness	When some nodes are compromised, the entire network **should not** also become compromised. The quantitative value with which this requirement should be satisfied depends on the application.
Self-organization	Nodes **should** be independent and flexible enough to be self-organizing (autonomous) and self-healing (failure tolerant).
Availability	The network **should not** fail frequently.
Time synchronization	Collaborative node applications **need** time synchronization. Time synchronization protocols **should not** be manipulated to produce inaccurate time.
Secure localization	Nodes **should** be able to accurately and securely acquire location information.

Table 28: Security requirements for a Wireless Sensor Network

For key management, the most critical requirements are *robustness* and *self-organization*. Although *confidentiality* and *integrity* are important, the ability to distribute secret, shared keys satisfies both requirements, and all key management schemes are able to accomplish this. Likewise, *data freshness* is typically attained by including a **nonce** (*a cryptographic time stamp*) in each packet to verify that the data is new. That approach hinges on the integrity of each data packet to ensure the nonce has not been modified, which easily can be accomplished after a secret, shared key is established. On the other hand, *self-organization* — the ability to independently self-organize and self-heal in the presence of dynamic changes in a WSN — is a requirement that is more difficult to satisfy. Normally, without the considerations of security, WSNs are designed to satisfy this property such that nodes can freely form connections around a failed node or re-establish the network automatically after it has been disturbed.

However, when a key management scheme distributes certain communication keys to a few nodes, this requirement can be violated as other nodes are unable to form connections dynamically with these specific nodes for the lack of proper keys.

With robustness, the problem lies with the compromise of one or more nodes. Because WSN nodes are frequently deployed in unsupervised and remote locations, physical tampering is a real threat, and the WSN must be able to withstand the compromise of some nodes. If the network uses only one key, then the compromise of one node compromises the entire network. If the network uses multiple keys, it is interesting to determine how many compromised nodes it takes to compromise the security of the entire network.

Moreover, several operational requirements exist for WSNs: *accessibility*, *flexibility*, and *scalability* (Table 29). These requirements act as constraints for security design because one must ensure that they are not violated in the design and realization of a security scheme. Accessibility, the need for data to be accessible by many nodes, arises as WSNs must efficiently utilize the limited energy, computation, and memory resources. A popular scheme to enhance resource efficiency is data aggregation or data fusion, which combines data from other nodes with local data before routing the fused data towards the sink node. This process

requires the intermediate nodes to interpret the data being sent by the other nodes. Likewise, at event detection points, neighbour nodes must be able to overhear transmissions from other nodes to ensure that only one node reports the event instead of every possible node.

Another crucial operational requirement is flexibility. For example, in a large WSN, nodes can frequently join or leave the network due to network expansion or battery depletion. A key management scheme must support this process with easy addition or deletion of keys. This is challenging as the lack of a central key management entity makes the addition and deletion of keys a tedious, inefficient, and more importantly, an insecure task. Lastly, the scalability of the key management scheme is an important performance factor. IEEE 802.15.4-compliant WSNs can theoretically support up to 65,536 nodes. Therefore, a key management scheme must be scalable up to this maximum network size.

Requirement	Description
Accessibility	Intermediate nodes **should** be able to perform data aggregation by combining data from different nodes. Neighboring nodes **should** also be able to passively monitor event signals to prevent large amounts of redundant event signaling information.
Flexibility	Nodes **should** be replaceable when compromised. On-the-fly addition of nodes should also be supported.
Scalability	A WSN **should** concurrently support at least 3000 nodes even with the key management scheme in place.

Table 29: Operational requirements for Wireless Sensor Networks

4.2.2 Key Distribution scheme

The three simplest keying models that are used to compare the different relationships between the WSN security and operational requirements are *network keying*, *pairwise keying*, and *group keying*. Table 30 summarizes a detailed problem and benefit analysis.

Model	Description	Benefits	Problems
Network	The entire network uses one shared secret key.	• Simple • Allows data aggregation and fusion • Scalable • Able to self-organize • Flexible/accessible	• Compromise of one node compromises the entire network (lacks robustness)
Pairwise	Each specific pair of nodes shares a different key.	• Best robustness • Authenticates each node	• Nonscalable — storage, energy, computation • Unable to self-organize • Not flexible for addition/removal of nodes
Group	Each group uses a different shared key.	• Allows multicast • Allows group collaboration • Better robustness than network-wide keying • Adjustable scalability • Addition/removal of nodes possible • Able to self-organize within the cluster	• Lacks efficient storage method for group keying in IEEE 802.15.4 • Difficult to set up securely • Cluster formation information is application-dependent

Table 30: Common Key Distribution Schemes for Wireless Sensor Networks

The *network keying model* has inherent advantages over the other two schemes. It is simple, easy to manage, and uses very little resources. It also allows easy collaboration of nodes since neighbouring nodes can read and interpret each other's data, satisfying the self-organization and accessibility requirements very well. It is also excellent in terms of scalability and flexibility because there is only one key for the entire network, and it does not change with the addition of nodes. However, an unacceptable drawback in robustness exists. Suppose one node is compromised, and the network wide key is exposed. With this key, an adversary can eavesdrop on all messages in the network and even inject forged messages into the network, possibly disrupting the proper operation of the network.

At the other extreme, the *pairwise keying model* employs $N - 1$ keys in each node, where N is the size of the network. Although this model provides the ultimate in robustness against node capture because the compromise of one node does not compromise any other node, it fails to satisfy the scalability requirement because the storage cost grows rapidly with network size. In the case of several thousand nodes, the number of keys each node must maintain becomes unmanageable. Consider the storage of $N - 1$ keys per node. The total number of distinguishable keys in the network is $N(N - 1)/2$, which grows at a rate ofN^2; this is not maintainable when N is a large value. Another issue with the pairwise keying model is that it is difficult to add new nodes to the network, affecting the flexibility requirement. When a new node is added, every node must obtain a new key to communicate with it. This is a resource-intensive process that uses much more precious energy when compared with the simple preloading of a network-wide key as in the previous model. Similarly, key revocation and key refreshing suffer from the same scalability problem. Additionally, the accessibility requirement is in jeopardy as nodes cannot passively monitor event signals.

Lastly, in the case of some pairwise key distribution schemes, self-organization comes into question, because they tackle the scalability problem by reducing the number of shared keys, resulting in some nodes being unable to communicate with others and compromising the self-healing and self-organizing abilities of the network.

The *group keying scheme* combines the features of both network and pairwise keying schemes. Within a group of nodes that form a cluster, communications are performed using a single, shared key similar to network keying. However, communications between groups employ a different key between each pair of groups in a manner identical to the pairwise keying scheme. Thus, for a group of nodes, the accessibility requirement is satisfied because data aggregation can occur with no additional cost while some degree of robustness is maintained. When one of the nodes is compromised, the worst-case scenario is the compromise of the entire cluster that it belongs to, which is considerably more isolated than the entire network. In terms of scalability, an acceptable trade-off is possible in this scheme, because the number of keys increases with the number of groups, not with the size of the network. However, the problem with this scheme is that it is difficult to set up and also, the formation of the groups is a very application dependent process. To efficiently distribute the keys, a keying scheme would require group formation information.

4.3 Traffic Analysis

Internet data on their way from source to the final destination go through many intermediary nodes and this is also the case for IoT. Most of these nodes are untrustworthy as they are neither in the control of the sender nor of the recipient. When Internet packets traverse these nodes they leave traces and leak information about the content of traffic they transport. It is widely believed and accepted that encryption (provided that it is secure and cannot be broken) hides the content of data traffic. This, however, is only partially true because by observing patterns in packet sizes, timing, etc. of various protocols, it is under certain circumstances still possible to infer information about the content even without breaking the encryption. Despite encryption, these patterns of communication (i.e., direction, frequency, size) are directly

visible to an eavesdropper and intermediate nodes. The situation becomes particularly severe in wireless settings that are typical for the IoT world: everybody in the receiving range of the signal is able to eavesdrop on the traffic to conduct traffic analysis. Hence, users of wireless networks such as those with mobile devices whose number is rapidly increasing and IoT devices are especially exposed to the threat of traffic analysis. This analysis is performed passively and therefore undetectable for the potential victims. Moreover, the wide penetration of easy-to-use hacking tools for different platforms makes it trivial even for non-experts to perform malicious activities. At the same time emerging use cases of the Internet such as smart metering and cloud services imply an increased transmission of private data through the network.

Traffic analysis works as follows. In the first step, the attacker extracts features from the observed data streams. In the second step, he applies machine learning techniques to identify patterns of communication. As a preparation for the attack, the attacker needs to learn the pattern of the content that he wants to recognize. This is performed by downloading the content and capturing the traffic in the same way as it is done with the traffic of the victim (but in this case the attacker knows what is being transmitted). The main challenge in this process is to define appropriate features that would allow a reliable distinguishing of the searched for pattern.

In data mining and machine learning, a feature (sometimes also called an attribute) is the smallest piece of information that is given to the classifier. The task of feature engineering is to find a set of attributes that make different classes (e.g., transferred objects, websites, words in case of a VoIP traffic) distinguishable, i.e., that are able to model characteristics of single classes. This leads to a high classification accuracy and performance. This is practically done by eliminating features with little or no predictive information while finding and including attributes with much predictive information. It is also a common strategy to make certain characteristics, already implicitly presented in the data, explicit to increase the accuracy of classification. Successful feature selection helps to drastically increase classification accuracy compared to the application of classifiers on the raw data. Feature engineering is an important step in this process as features allow expressing characteristics of the data.

Already back in 1996, despite the success of the encryption, Wagner and Schneier pointed out that traffic analysis could be used to draw conclusions about content of encrypted SSL packets. Two years later, the attack was successfully mounted against a single server offering different URLs. Several early works analysed traffic flow confidentiality assuming that every web object can be differentiated, which does not hold true in pipelining protocols used nowadays (the original attack was demonstrated with the HTTP/1.0protocol). Later works studied a small scale of closed-world scenario only, exemplary web applications protected by SSL, managed to gain only some of the information, e.g., the language of a VoIP call. Some other researchers even managed to perform traffic analysis remotely, without having a direct contact to the communication link. This is a big threat as the attacker can launch his attack without ever coming into physical proximity with the victim. This is done by identifying a queuing side channel that can be used to infer the queue size of a given link with good accuracy. It makes the attacker able to remotely monitor traffic patterns.

Today, the limits of traffic analysis in presence of state-of-the-art protection techniques are still not well studied. At the same time its risks are especially severe in mobile networks as everyone in the vicinity of ascending node is capable of eavesdropping on the traffic. Furthermore, untrustworthy intermediary nodes -- that are naturally involved in relaying data packets also have the power to capture traffic flows. The vulnerability of existing privacy-preserving communication protocols using the example of anonymization networks Tor [76] and JAP [77] is analysed.

Anonymization networks such as Tor and JAP promise anonymity by routing data through

several overlay nodes and using layered encryption of the content. They promise to strengthen a user's civil rights, to protect the privacy, or even to give a user the opportunity to evade the censorship. The vulnerability of these networks w.r.t traffic analysis using the example of website fingerprinting is analysed. Website fingerprinting (WFP) is an attack that can be performed by a local attacker (e.g., a local system administrator), an ISP, or anybody who is able to monitor the link between the original message sender and the first node of the anonymization network (e.g., everyone in the sending range of a victim communicating via a wireless link). Local eavesdropping is one of the weakest attacks one can imagine. WFP is a special form of traffic analysis where a local attacker observes the encrypted data and tries to draw conclusions from certain features of the traffic, such as the volume of transferred data, the timing or the packet sizes. The method does not aim at breaking the cryptography, but even if an attacker does not understand the message semantics, he can try to match observed patterns to known patterns of, e.g., web pages. If successful, website fingerprinting destroys the whole protection and anonymity promised by, e.g., Tor and JAP as it can be carried out by local attackers.

In practice, an attacker first retrieves a certain amount of relevant web pages by himself as training data for fingerprinting, using the anonymization network that he assumes his victim uses as well. He records the transferred packets with a traffic analyser tool such as tcpdump which provides information about IP layer packets, e.g., the length of the packet, the time the packet was sent or received, the order in which the packets were sent and received, etc. The attacker can use various information contained in the dumps to create a profile for each web page, the so-called fingerprint. Later, wiretapping on the victim's traffic, the attacker similarly collects data that we call test data. The test data resembles the fingerprints, but it usually differs from them due to a number of reasons, e.g., non-deterministic packet fragmentation, updates in web pages, etc. Hence, the attacker needs to apply statistical methods to compare the recorded information to the fingerprints and to probabilistically match it to a certain web page.

In our research, an attacker's practical procedure is stimulated. In doing so, our methods were evaluated for fingerprinting and countermeasures by collecting two different data sets. The Closed-World Dataset is taken from the recent related work of Herrmann et al. [75] in order to compare our results to the best results achieved previously. Based on this data set, we show superiority of our features and methods compared to the state-of-the-art work. Next, this proof-of-concept scenario was extended which uses a closed world assumption vs. an open world one. To make it realistic, 1,000,000 most popular Internet pages were used from a well-known web statistics service to collect our Open-World Dataset.

Previous works on website fingerprinting only used the packet size and whether packets are incoming or outgoing as features. Additional features were developed that help to heavily improve the detection rates both in closed and open-world scenarios. Clearly, feature engineering is far from trivial. Since the power of features cannot be predicted easily, we empirically tested a large set of features.

Using Bayes Nets, Herrmann et al. [75] achieve a recognition rate of 3% on the Tor Closed-World Dataset. The algorithm of Herrmann as described in [75] was implemented and achieved similar results on the dataset. Are cognition rate of less than 3% might appear non-threatening for many users? The use of Support Vector Machines (SVM) instead of Bayes Nets as the first major expansion was proposed. The more powerful SVMs directly lead to an improved recognition of almost 31%. By considering the proposed features, the recognition rate was increased to 47.36%, an increase by 17%. The proposed algorithm improves the previously known detection rate of Herrmann by more than 51 %. A detection rate of almost 55% on average is alarming. The anonymization using Tor is far less robust as previously assumed. Our improved methods for website fingerprinting on the JAP Closed-World Dataset was also applied. Herrmannet al. [75] achieved a detection rate of 20% in the JAP network.

Their study is limited to a premium cascade only. Using Bayes Nets, similar results of 26% in a premium cascade was achieved. With our algorithm, the detection rate was boosted to almost 80%. Our results confirm that the JAP framework also does not provide the anonymity as previously thought. In addition, the accuracy of recognition in free cascades was investigated . Using Bayes Nets, 3.5% only was achieved. The application of our approach leads to a recognition rate of 60%. The overall result is surprising because JAP operators claim that premium cascades (paid) offer more anonymity to their users than free cascades. The surprising effect can be explained by fewer users in premium cascades due to higher costs. This fact leads to a lower workload. As further investigation experiments confirm, less workload directly leads to an increase in recognition rate of about 30%. For both cascade types in JAP, higher detection rates compared to Tor were achieved. Therefore, JAP seems to be more vulnerable against fingerprinting in the considered scenario. Similar to Tor, the selected features play an important role for the high detection rates. Both for premium and free cascades, the features boost the results by at least11%. The high recognition rates on the Closed-World Dataset confirm the generality and importance of the selected features and our method for the whole domain of website fingerprinting. This indicates the need to take appropriate counter measures.

5 QoS

After a first analysis, it has been decided that IoT6 QoS contributions are related to smart routing. Therefore, the corresponding contribution is neither paraphrased nor split between two deliverables and we encourage the reader to read D3.2.

6 Conclusions

A number of key technologies to enhance the security and privacy aspects of the upcoming Internet of Things were presented in this deliverable with two main categories of technologies described. On the one hand, technologies that can be quickly operational in order to address the basic security and privacy requirements of services running on top of the IoT6 architecture were studied. To this end, well established technologies which have been proven compatible with 6lowpan and 802.15.4 like CCM, DTLS, LSEND, etc. have been investigated and advices have been given on their usage, mainly in sections 3.1 and 4. Also, beyond security considerations, privacy aspects have not been neglected and different ways to use encryption techniques, with proper key management, and traffic analysis were studied.

On the other hand, we also tried to push forward our work in security by considering more advanced problematics that may arise in the future of IoT and which lead us to investigate more pervasive and research-oriented technologies. In this context, we proposed an advanced large scale monitoring solution featuring multidimensional aggregation which can be applied to IPV6 flows. A new set of security features to secure communications based on the Content-Centric Networking paradigm, which includes a firewall and a key management scheme, and from which IoT communications could highly take advantage was also proposed .

Our future work will consist in applying the selected security and privacy technologies within the IoT6 platform so that the future IoT services can benefit seamlessly from them.

7 List of Acronyms

6LoWPAN: IPv6 Low power Wireless Personal Area Network
AES: Advanced Encryption Standard
AES-CCM: Advanced Encryption Standard Counter with CBC-MAC
AH: Authentication Header
ANSI: American National Standards Institute
CBC: Cipher Block Chaining
CBC-CS: Cipher Block Chaining Ciphertext Stealing
CBC-MAC: Cipher Block Chaining Message Authentication Code
CIDR: Classless Inter-domain routing
CMAC: Cipher-based Message Authentication Code
CTR: Counter
CCM: Cipher Block Chaining Mode
CPU: Central Processing Unit
DDOS: Distributed Denial of service
DES: Data Encryption Standard
DNS: Domain Name System
DTLS: Datagram Transport Layer Security
ECAES: Elliptic Curve Augmented Encryption Scheme
ECC: Elliptic Curve Cryptography
ECDH: Elliptic Curve Diffie Hellman
ECDLP: Elliptic Curve Discrete Logarithmic Problem
ECDSA: Elliptic Curve Digital Signature Algorithm
ECIES: Elliptic Curve Integrated Encryption Scheme
ESP: Encapsulating Security Payload
FFD: Full Function Device
FIFO: First In First Out
HAA: Hardware Abstraction Architecture
HIL: Hardware Interface Layer
HMAC: keyed-hash Message Authentication Code
ICV: Integrity Check Value
IdM: Identification Management
IETF: Internet Engineering Task Force
IKE: Internet Key Exchange
IoT: Internet of Things
IPSec: Internet Protocol Security

8 References

[1] L. Atzori, A. Iera and G. Morabito, "The Internet of Things: A survey". Computer Networks Vol. 54, No. 15, pp. 2787-2805, 2010.

Postel J. User datagram protocol. RFC768, Internet Engineering Task Force; August 1980

[2] Z. Shelby and C. Bormann, "6LoWPAN: The Wireless Embedded Internet", Wiley, 2009.

[3] J.P. Vasseur, Adam Dunkels, "Interconnecting Smart Object with IP", Morgan Kaufmann

[4] R. Watro, D. Kong, S.Cuti, C. Gardiner, C. Lynn and P. Kruus, "TinyPK: Securing Sensor Networks with Public Key Technology", Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks SASN'04, 2004.

[5] T. Reusing, "Comparison of Operating Systems TinyOS and Contiki", Proceedings of the Seminar Sensor Nodes – Operation, Network and Application (SN), Summer Semester 2012, volume NET-2012-08-2 of Network Architectures and Services (NET), pages 41-46, Munich, Germany, August 2012.

[6] 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 Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI), ACM, 2003.

[7] A. Dunkels, B. Gronvall, T. Voigt Contiki, "a Lightweight and Flexible Operating System for Tiny Networked Sensors" In Proceedings of the First IEEE Workshop on Embedded Networked Sensors, Tampa, Florida, USA, 2004.

[8] C. Karlof, N Sastry and D. Wagner, "TinySec: A Link Layer Security Architecture for Wireless Sensor Networks" In proceedings of the International Conference ACM SENSYS'04, Baltimore, Maryland, November 3-5, 2004.

[9] N. Koblitz, "Elliptic curve cryptosystems", Mathematics of Computation 48, (177): 203-209, 1987.

[10] V. Miller, "Use of elliptic curves in cryptography", CRYPTO 85: 417-426, 1985.

[11] A. Liu and P. Ning, "TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Network" In Proceedings of the 7th International Conference on Information Processing in Sensor Networks (ISPN 2008), SPOTS Track, p245-256, April 2008.

[12] Certicom Research, "Standards for efficient cryptography - SEC1: Elliptic curve cryptography", http://www.secg.org/download/aid-385/sec1_final.pdf, September 2000.

[13] Certicom Research, "Standards for efficient cryptography – SEC2: Recommended elliptic curve domain parameters", http://www.secg.org/collateral/sec2_final.pdf, September 2000.

[14] L. Casado and P. Tsigas, "ContikiSec: A Secure Network Layer for Wireless Sensor Networks under the Contiki Operating System" In proceedings of the 14th Nordic Conference on Secure IT Systems, Oslo, Norway, 2009.

[15] S. Raza, T. Chung, S. Duquennoy, D. Yazar, T. Voigt and U. Roedig, "Securing Internet of Things with Lightweight IPSec", SICS Technical Report, 2012.

[16] S. Raza, T. Voigt and V. Jutvik, “Lightweight IKEv2: A Key Management Solution for both Compressed IPSec and IEEE 802.15.4 Security”, In IETF Workshop on Smart Objects Security, Paris, France, 23 March 2012.

[17] V. Perelman, “Security in IPv6-enabled Wireless Sensor Networks: An Implementation of TLS/DTLS for the Contiki Operating System”, MSc Thesis, Jacobs University Bremen, August 2012.

[18] O. Bergmann, “Tinydtls”, http://tinydtls.sourceforge.net.

[19] IEEE standard for Information Technology, "IEEE std. 802.15.4, Part. 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks", June 2011.

[20] S. Park, K. Kim, W. Haddad, S. Chakrabarti, J. Laganier, IPv6 over Low Lower WPAN Security Analysis, draft-daniel-6lowpan-security-analysis-05, Internet Engineering Task Force , March 2011.

[21] Dolev D. and Yao A., On the security of publick key protocols. *In Proc. of the IEEE 22nd Annual Symposium on Foundation of Computer Science*, pp. 350-357,1981.

[22] Rescorla E., and Korver E., *Guidelines for Writing RFC Text on Security Considerations*. RFC3552, Internet Engineering Task Force, July 2003.

[23] S. Kent, and K. Seo, *Security Architecture for the Internet Protocol*. RFC4301, Internet Engineering Task Force,December 2005.

[24]D. Harkins, D. Carrel, *The Internet Key Exchange (IKE).* RFC2409, Internet Engineering Task Force, November 1998.

[25]C. Kaufman, *Internet Key Exchange (IKEv2) Protocol*,RFC4306, Internet Engineering Task Force, December 2005.

[26] K. Stammberger, M. Semp, M.B. Anand, and D. Culler “Introduction to Security for Smart Object Networks”, Internet Protocol for Smart Objects (IPSO) Alliance, White paper #4, February 2010.

[27] S. Kent, *IP Encapsulating Security Payload (ESP)*. RFC4303, Internet Engineering Task Force, December 2005.

[28]S. Frankel, R. Glenn, S. Kelly, *The AES-CBC Cipher Algorithm and Its Use with IPSec*, RFC 3602, Internet Engineering Task Force, September 2003.

[29]R. Housley, *Using Advanced Encryption Standard (AES) CCM Mode with IPSec Encapsulating Security Payload (ESP)*. RFC4309, Internet Engineering Task Force, December 2005.

[30] T. Dierks , and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*.RFC5246. Internet Engineering Task Force, August 2008.

[31]Gupta V, Wurm M , Zhu Y , et al . Sizzle: A standards-based end-to-end security architecture for the embedded internet. *Pervasive Mobile Comput J* . December 2005: 425 – 445.

[32]Oliveira L, Kansal A, Priyantha B, Goraczko M, Zhao F. Secure-TWS: Authenticating node to multiusercommunication in shared sensor networks. In: *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks, IPSN 2009*; 2009:289 – 300.

[33] J. C. Lee, V.C.M. Leung, K.H. Wong, J.Cao, and H.C.B. Chan, “Key Management Issues in Wireless Sensor Networks: current proposal and future developments”, IEEE Wireless Communications, October 2007.

[RFC6347] E. Rescorla and N. Modadugu. Datagram Transport Layer Security (DTLS) Version 1.2. RFC 6347 (Proposed Standard), January 2012.

[34] C. Bormann. *6LoWPAN Generic Compression of Headers and HeaderlikePayloads*.draft-bormann-6lowpan-ghc-04, March 2012.

[35] Z. Shelby, K. Hartke, C. Bormann, and B. Frank. Constrained ApplicationProtocol (CoAP), March 2012.

[36] S. Raza, D. Trabalza, T. Voigt, *6LoWPAN Compressed DTLS for CoAP*, in Proc. of the 8th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS), Hangzhou, China, 2012.

[37]M. Brachmann, S. L. Keoh, O. G. Morchon, and S. S. Kumar, *End-to-End Transport Security in the IP-basedInternet of Things*, in Proc. of 21st International Conference on Computer Communications and Networks (ICCCN), Munich, Germany, 2012.

[38] R. Hinden, S. Deering, RFC 4291, "IP Version 6 Addressing Architecture", February 2006

[39] B. Claise, G. Adasivan, Valluri V., M. Djernaes, Rfc 3954 “Cisco systems netflowservices export version 9”, 2004.

[40] C. Wagner, J. François, R. State, T. Engel, “Danak: Finding the odd!”, In Network and System Security (NSS), 2011 5th International Conference on (2011), IEEE, pp. 161–168.

[41] V. Levenshtein, “ Binary codes capable of correcting deletions, insertions, and reversals”. In Soviet physics doklady (1966), vol. 10, pp. 707–710.

[42] P. Bille, “A survey on tree edit distance and related problems. Theoretical computer science 337, 1 (2005), 217–239.

[43] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Br iggs, and Rebecca L. Bray nard. Networking named content. In Proceedings of the 5th international con ference on Emerging networking experiments an d technologies, CoNEXT '09, pages 112, New York, NY, USA, 2009. ACM.

[44] Lixia Zhang, Deb orah Estrin, Jerey Burke, Van Jacobs on , James D. Thornton, Diana K. Smetters, Beichuan Zhang, Gene Tsudik, kc clay, Dmitri Krioukov, Dan Massey, Christos Papadopoulos, Tarek Abdelzaher , Lan Wang, Patrick Crowley, and Edmund Yeh. Named Data Networking (NDN) Project, October 2010.

[45] S. DiBenedetto, P. Gasti, G. Tsudik, and E. Uzun, “Andana: Anonymous named data networking application,” CoRR, vol. abs/1112.2205, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1112.html#abs-1112-2205

[46] T. Lauinger, “Security & scalability of content-centric networking,”September 2010. [Online]. Available: http://tubiblio.ulb.tu-darmstadt.de/46912/

[47] D. K. Smetters and V. Jacobson. Securing network content, October 2009.PARC Technical Report.

[48] CCNx Project webpage, http://www.ccnx.org/wiki/working-with-ccnx-android-code/

[49] Bilel Saadallah, Abdelkader Lahmadi, Olivier Festor, “Mise en oeuvre d’une couche de communication CCN pour les réseaux de capteurs sans fil”. Mémoire d’Ingénieur, ENSI-TUNISIE, 2011.

[50] S. Y. Oh, D. Lau, and M. Gerla. Content Centric Networking in tactical and emergency MANETs. In Wireless Days (WD), 2010 IFIP, pages 1{5. IEEE, October 2010.

[51] L. Zhou and Z. Haas, “Securing ad hoc networks,” Network, IEEE ,vol. 13, no. 6, pp. 24

–30, nov/dec 1999.

[52] J. Kong, Z. Petros, H. Luo, S. Lu, and L. Zhang, “Providing robust and ubiquitous security support for mobile ad-hoc networks,” in Network Protocols, 2001. Ninth International Conference on, nov. 2001, pp. 251–260.

[53] H. Luo, P. Zerfos, J. Kong, S. Lu, and L. Zhang, “Self-securing adhoc wireless networks,” in Computers and Communications, 2002.Proceedings.ISCC 2002. Seventh International Symposium on , july 2002, pp. 567 – 574.

[54] S. Capkun, L. Buttyan, and J.-P. Hubaux, “Self-organized public-key management for mobile ad hoc networks,” Mobile Computing, IEEE Transactions on, vol. 2, no. 1, pp. 52 – 64, jan.-march 2003.

[55] A. Khalili, J. Katz, and W. Arbaugh, “Toward secure key distribution in truly ad-hoc networks,” in Applications and the Internet Workshops, 2003.Proceedings. 2003 Symposium on, jan. 2003, pp. 342 – 346.

[56] H. Deng, A. Mukherjee, and D. Agrawal, “Threshold and identity-based key management and authentication for wireless ad hoc networks,” in Information Technology: Coding and Computing, 2004. Proceedings.ITCC 2004. International Conference on , vol. 1, april 2004, pp. 107 –111 Vol.1.

[57] D. R. G. Rossini, “Caching performance of content centric networks under multi-path routing (and more),” in Technical report, Telecom ParisTech, 2011.

[58] Cristianini, N., Shawe-Taylor, J.: An introduction to support Vector Machines and other kernel-bsed learning methods. Cambridge University Press, New York,USA (2000)

[59] Debnath, R., Takahide, N., Takahashi, H.: A decision based one-against-one method for multi-class support vector machine. Pattern Anal. Appl. 7(2), 164-175 (2004)

[60] Wang, L. (ed.): Support Vector Machines: Theory and Applications, Studies in Fuzziness and Soft Computing, vol. 177. Springer (2005)

[61] CCNDSMOKETEST Manual Page, http://www.ccnx.org/releases/latest/doc/manpages/ccndsmoketest.1.html

[62] D. H. Crocker and P. Overell, “Augmented bnf for syntax specifications: Abnf,” Internet RFC 4234, 2005.

[63]T. Segaran and J. Hammerbacher, Beautiful Data: The Stories Behind Elegant Data Solutions. O’Reilly Media, 2009, ch. 14.

[64]P. Kolb, “DISCO: A Multilingual Database of Distributionally Similar Words,” in KONVENS 2008 – Erg¨anzungsband: Textressourcen und lexikalisches Wissen, 2008.

[65] N. Alexiou, S. Basagiannis, P. Katsaros, T. Dashpande, and S. A. Smolka, “Formal analysis of the kaminsky DNS cache-poisoning attack using probabilistic model checking,” in International Symposium on High-Assurance Systems Engineering (HASE). IEEE, 2010.

[66] D. Atkins and R. Austein, “RFC 3833: Threat Analysis of the DNS,” 2004.

[67] J. François, S. Wang, R. State, and T. Engel, “Bottrack: tracking botnets using netflow and pagerank,” in NETWORKING 2011. Springer Berlin Heidelberg, 2011, pp. 1–14.

[68] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling, “Measuring and detecting fast-flux service networks,” in Network and Distributed System Security Symposium (NDSS), 2008.

[69] S. Marchal, J. François, R. State, and T. Engel, “Proactive discovery of phishing related domain names,” in Recent Advances in Intrusion Detection, ser. LNCS. Springer, 2012.[Online].

[70] R. Perdisci, I. Corona, and G. Giacinto, “Early detection of malicious flux networks via large-scale passive DNS traffic analysis,” Transactions on Dependable and Secure Computing, pp. 714–726, 2012.

[71] S. Marchal, J. François, C. Wagner, R. State, A. Dulaunoy, T. Engel, and O. Festor, “DNSSM: A large-scale Passive DNS Security Monitoring Framework,” in IEEE/IFIP Network Operations and Management Symposium, 2012.

[72] F. Weimer, “Passive DNS replication,” 2005.

[73] L. Dolberg, J. François, and T. Engel, “Multi-dimensional Aggregation for DNS Monitoring“, to appear in the proceeedings of the 38th IEEE Conference on Local Computer Networks (LCN), 2013

[74] L. Dolberg, J. François, and T. Engel Efficient Multidimensional Aggregation for Large Scale Monitoring”, in USENIX Large Installation System Administration Conference (LISA), 2012

[75] D. Herrmann, R. Wendolsky, and H. Federrath. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Na ̈ive-Bayes Classifier. In CCSW ’09: Proceedings of the 2009 ACM CCS Workshop on Cloud Computing Security, pages 31–42, Chicago, Illinois, USA, Nov. 2009. ACM Press.

[76] R. Dingledine, N. Mathewson, and P. Syverson, “Tor: The Second-Generation Onion Router,” in Proceedings of the 13th USENIX Security Symposium. USENIX Association, Aug. 2004, pp. 303–320.

[77] O. Berthold, H. Federrath, and S. Köpsell, “Web MIXes: A System for Anonymous and Unobservable Internet Access,” in Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability, ser. Lecture Notes in Computer Science, H. Federrath, Ed., vol. 2009. Springer-Verlag, Jul. 2000, pp. 115–129.

[78] Postel J. “User datagram Protocol”. RFC768, Internet Engineering Task Force; August 1980.

[79] Z. Shelby, S. Chakrabarti, E. Nordmark, Neighbor Discovery Optimization for Low Power and Lossy Networks (6LoWPAN) draft-ietf-6lowpan-nd-21, August 2012.