An All-Round Secure IoT Network Architecture

Master Thesis

Author(s):
Tulimiero, Andrea

Publication date:
2020

Permanent link:
https://doi.org/https://doi.org/10.3929/ethz-b-000443426

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Master Thesis

Network Security Group, Department of Computer Science, ETH Zurich

An All-Round Secure IoT Network Architecture

by Andrea Tulimiero

Spring 2020

ETH student ID: 18-947-150
E-mail address: tandrea@ethz.ch

Supervisors: Piet De Vaere
Prof. Dr. Adrian Perrig

Date of submission: September 11, 2020

Abstract

The adoption of the Internet of Things (IoT) is growing steadily and is increasingly becoming more and more widespread every day. As a consequence, we rely on IoT appliances to perform every kind of task: from turning on and off a light with a voice command, to remotely control medical equipment. On the one hand, IoT can hugely improve our lives, but on the other hand, its adoption opens up to a whole new class of threats. It is then paramount that companies and IoT producers consider security a top priority in the realization of their products. However, several recent IoT related security incidents revealed that the security of these devices is sometimes completely ignored. Further inspecting these incidents, we discover that the security of IoT devices shall be achieved first and foremost in the network, which is, at the same time, its major enabler and security weakness. There is a clear need for a system that offers high-security guarantees while posing the lowest effort possible on IoT manufacturers, so to help them adopt security best practices in their production processes. Instead of starting from the network, our analysis follows a bottom-up approach by first systematically defining what security capabilities an IoT environment should meet, as suggested in security guidelines and frameworks from both industry and academia. Then, we realize a network architecture that offers the required services to build a secure IoT environment. The resulting architecture is a combination of LaNeCa and SCION, an intra- and an inter-domain protocol, respectively. The solution, implemented with compatibility and performance in mind, can be integrated by vendors into their products with a minimum effort.

Acknowledgements

I would like to start this thesis by thanking the people that made it possible. First, I want to thank my co-supervisor, Piet, who mentored me step-by-step during this challenging journey. His help and his knowledge have been truly inspirational. Second, I want to thank my supervisor, Prof. Adrian, for giving me the chance to work on this exciting topic and for pushing me to constantly improve my work. I also want to thank Juan and all the people of the Network Security Department for their availability to answer my questions, and Peter for its American English advice. Finally, I want to thank my family, who immensely supported me for these last two years and gave me the chance to chase my dreams.

Contents

1 Introduction

The Internet of Things (IoT) is "a system of entities (including cyber-physical devices, information resources, and people) that exchange information and interact with the physical world by sensing, processing information, and actuating" [IEE20b]. The IoT has surpassed its 'buzzword phase' common to any new game-changing technology, and it is now common to find IoT devices in our everyday life. The adoption of IoT is growing steadily [Blo20, Gar20b] and its audience is broadening. For instance, the employment of IoT by enterprises gave birth to what is called industry 4.0[Gar20a, SOM14].

The major enablers that ensured the seamless interconnection of billions of pervasive embedded systems are two: (i) a more and more accessible Internet, both in terms of price and availability and (ii) an unprecedented level of miniaturization of electronic components [OMM17]

1.1 Current State of IoT (in)security

Internet connection, constrained resources, and companies rushing to deliver products on the market [Tim20] made it impossible to properly secure IoT devices. Traditional security solutions, which can offer manufacturers a plug-and-play base of security for their products, are, in most cases, incompatible with IoT devices [Sym20, TC20]. IoT manufacturers were then supposed to develop new lightweight, tailor-made security systems for their IoT products to meet appropriate privacy and security requirements. Unfortunately, companies were not motivated enough to consider the security of their products a top priority in their production process [Axe15]. Such superficial management of security paved the way for countless opportunities of cyberattacks, which were promptly seized not only by security researchers but also from cybercriminals.

Everything is getting compromised: (i) smart home door locks [O'D19] and security cameras [Gá19], (ii) industrial machines [QPP+17], (iii) medical anesthesia appliances [Bar19] and insulin pumps [Tor19], and (iv) military strykers [Tre19]. As if the scenario was not scary enough, vulnerable IoT devices represent a threat not only to themselves and the physical world around them but also to worldwide connectivity.

One reason IoT devices are particularly appealing to cyber criminals is that they are the perfect target for botnets [New20]. They are pervasive, always connected to the Internet, and easily hijackable in clusters. This last characteristic is especially dangerous: because of their hardware, software, and configuration homogeneity, once a model of IoT device gets compromised, any other similar model can be compromised, and being all Internet-connected, they indeed get all compromised. The Mirai [AAB+17] botnet, one of the most severe distributed denial-of-service (DDoS) attacks ever registered [Clo17], showed the world that these are not just hypotheses.

1.2 Why Securing IoT Devices at Network Level?

The biggest threat of IoT is hiding in plain sight in its name: the Internet. The majority of attacks listed in Section 1.1 are indeed remotely performed over the Internet. By allowing required communications only, we can vastly reduce the attack surface exploitable by attackers. The Mirai botnet itself has been made possible by devices shipped with default credentials and opened ports. A more fine-grained network access control system would have been enough to spare the world one of its worst global connectivity outages.

1.3 Overview of Our Modus Operandi

We started by assessing the current state of IoT adoption and (in)security to clarify and highlight the need for this work. Chapter 2 covers the main concepts of the two technologies our solution is based on, giving the reader a primer (or a refresher) to help them understand better the following chapters. Chapter 3 defines what security services a network must offer to an IoT device so that it can meet its required security capabilities. We define these capabilities by reviewing security frameworks and guidelines aimed at IoT security. Then, with a bottom-up approach, we shape the security services a network must offer. At the end of Chapter 3, we select an intra- and inter-domain network technology to assemble our system. The selection process is performed by systematically analyzing: (i) the fulfillment of the security services a network must offer, and (ii) the compatibility with domain-specific requirements. For the remainder of this document, with the word 'network' we mean not only the network infrastructure used by a device, but also its network stack. Chapter 4 covers the implementation details of L2.5 LaNeCa, the chosen intra-domain protocol, investigating its integration in the network stack, and the realization of the needed cryptographic toolbox. At the end of Chapter 4, we also define a LaNeCa controller to facilitate the deployment and the management of a LaNeCa network. Chapter 5 explains how SCION, the chosen inter-domain protocol, is used to relay LaNeCa traffic across domains, and how we realize additional security services on top of it. Finally, in Chapter 6, we evaluate our system from multiple perspectives:

1. from a security perspective: analyzing what would be the consequences of the most promising attacks an adversary could unleash,

2. from a performance perspective: evaluating the overhead of Layered-independent Network Capabilities (LaNeCa), and

3. from a deployability perspective: assessing what are the challenges and advantages of operating a geographically distributed SCION augmEnted LaNECA (SENECA) IoT environment.

2 Background

This chapter is a primer, or a refresher, on the technologies used to build our architecture.

2.1 LaNeCa: Capability-Based, Deny-by-Default Networking

We start by introducing LaNeCa, a deny-by-default, capability-based, intra-domain network protocol. Being a deny-by-default protocol, LaNeCa allows only explicitely permitted traffic to circulate (all the rest is dropped), and achieves this by cryptographically verifying that each packet is part of an authorized flow. The verification is performed at every hop so that unauthorized packets get dropped as early as possible. The advantage of dropping packets early is twofold: first, devices get shielded from volumetric DOS attacks; second, it prevents the network fabric from overloading, which could raise latency and make the system unusable for some applications (e.g., industrial loops). The per-packet verification step involves no control plane traffic after the initial setup. This zero-communication verification of packet legitimacy is made possible by a capability mechanism based on a hierarchical key space. Authorized senders use capability tokens to generate authentication tags for each transmitted packets. These authentication tags get verified using hop's local only knowledge.

The environments targeted by LaNeCa share the following characteristics:

1. *ownership centrality*: which means that an IoT deployment is managed by a single entity (e.g., devices in a smart factory), and
2. *orchestration centrality*: which implies that although data flows can be distributed, these are still centrally orchestrated (e.g., the machines in a smart factory are configured by the factory management to execute a specific set of tasks, which lead to uniform and predictable communication patterns).

2.1.1 Hierarchical Capability Tokens

The capability of a device to send packets belonging to a flow translates into owning a set of keys called *flow key set*. This set of keys serves for the generation of a sequence of message authentication codes (MACs) that form a tag. The size and the number of MACs composing a tag can be changed to adapt to several use cases. A flow key set derives from a *receiver* key set, which, in turn, derives from a *root* key set. These sets of keys, which form a forest-like structure, compose the hierarchical key space of a LaNeCa installation. The construction of a key space starts from n root keys. Each i-th root key is a random secret, denoted k_i. The set of all root keys, the root key set,

$$\mathcal{K} = \{k_i \mid i \in \{1, \ldots, n\}\} \tag{2.1}$$

is fully known to the network controller only. Subsets of $\mathcal{K}$ get distributed to the switches in the network. For each tag, any switch in the network is thus able to check a subset of its MACs. Still, owning only a subset of the root keys, the switches are not able to forge valid packets on behalf of other entities. From $\mathcal{K}$, the receiver key set for a receiver r

$$\mathcal{K}^r = \{k_i^r = \mathrm{PRF}_{k_i}(r) \mid i \in \{1, \ldots, n\}\} \tag{2.2}$$

can be derived and dispatched to it to verify incoming packets. PRF_{k_i} is a pseudo-random function (PRF) keyed with k_i[1]. Finally, for a node s to communicate with host r via a protocol p, a flow key set can be derived from $\mathcal{K}^r$

$$\mathcal{K}^{r \leftarrow s:p} = \{k_i^{r \leftarrow s:p} = \mathrm{PRF}_{k_i^r}(s \| p) \mid i \in \{1, \ldots, n\}\} \tag{2.3}$$

and given to node s ("$\|$" is a string concatenation).

2.1.2 Security Equivalence Classes

For LaNeCa to scale, we introduce the concept of security equivalence classes. This concept asserts that since IoT environments are highly homogeneous, it is likely that if one device can get hacked, they can all get hacked. This observation is in line with what we highlighted in Section 1.1, being also one of the reasons that make IoT targets so appetible.

When analyzing the security of IoT networks, we then consider that either all the devices in a class are compromised or than none are. As a consequence, LaNeCa can withstand not only up to t compromised devices but rather, up to t compromised equivalence classes.

2.2 SCION: Next-Generation Internet

Scalability, Control and Isolation on next-generation Networks (SCION) is a clean-slate Internet architecture, aimed at scalability and security.

[1]In practice, block-cipher encryption can be used to implement the PRF

2.2.1 Isolation Domains

Like the Internet, SCION inherits the concept of autonomous systems (ASes). ASes in SCION are much more lightweight, enabling modest entities to run their own. ISolation Domains (ISDs) group ASes with distinct policies and interconnect to provide global connectivity. This added level of grouping, compared to regular Internet, makes it possible to:

- protect ASes from foreign ISDs routing attacks and misconfiguration,
- give endpoints control on both inbound and outbound traffic, and
- scale routing updates globally while keeping high path freshness.

Each ISD owns a separate Trust Root Configuration (TRC), which bootstraps all the cryptographic operations in it. In every ISD, a group of ASes, called core ASes, is in charge of its management.

2.2.2 Packet-Carried Forwarding State

The forwarding paradigm used by SCION is packet-carried forwarding state (PCFS): end hosts learn about available network path segments and combine them into end-to-end paths, which are encapsulated in packet headers and are used to steer the packet toward its destination. One of the advantages introduced by PCFS is multi-path communications. Thanks to multi-pathing, the routing infrastructure of SCION is remarkably agile, and it can quickly adapt to cope with congestion and link failures.

2.2.3 Pathing in SCION

A node that wants to send a packet over the SCION network needs a path to route it. Path construction covers a big chunk of routing in SCION, and every path is cryptographically signed to ensure its legitimacy. A path is a combination of up to three types of segments, which are, in turn, a sequence of ASes hops. The segment types are:

1. *up-segment*: which is a sequence of hops from a regular AS to a core AS,
2. *core-segment*: which is a sequence of hops from a core AS to another core AS, and
3. *down-segment*: which is a sequence of hops from a core AS to a regular AS.

Paths are not required to include every segment type and do not necessarily have to traverse all the hops contained in the segments (short-circuiting).

In the beginning, no paths exist, and each AS is connected to other ASes via direct links. These links can be of different types: core, parent-child, and peer. Core links are between core ASes. A parent-child link go from a core to a non-core AS, or between two non-core ASes, and, although not mandatory, indicates a provider-customer relationship. Peering links are similar to the Internet's equivalent. They are not involved in the beaconing process (which we will cover in the next paragraph) and can cross ISD boundaries – which is not possible for regular links due to the ISD routing isolation. Paths discovery happens with two processes: *beaconing*, for paths exploration, and *registration* for public path sharing. The beaconing process starts from the core ASes, who periodically propagate path-segment construction beacons (PCBs) to their children. Every non-core AS augments the received beacons with its information and keeps propagating them to its children, continuing with the path exploration process. The received PCBs are also converted in up-segments and stored in a local path server, to allow local entities to reach core ASes. ASes also register down-segments, derived like up-segments from the received PCBs, to inform other ASes how to be reached.

3 Towards a Comprehensive IoT Security Architecture

3.1 Defining Network Security Features

We define a secure network infrastructure with the following order: first, we outline the adversary we contrast; second, we analyze which security capabilities an IoT device should possess; and finally, we realize a network infrastructure that offers the required services to build a secure IoT environment.

3.1.1 Adversary Model

Goals

The goals of the modeled attacker are the following:

1. cause a *denial-of-service (DoS)* by making devices unreachable,
2. *steal* sensitive data from devices, either with an attack-and-exfiltrate approach or by intercepting it when traversing the network,
3. *corrupt* and damage a device or the world surrounding it, or both, exploiting its actuators, and
4. *exploit* a device to move laterally in the network to reach more appealing targets.

Capabilities

The modeled attacker can tamper with the network in any way possible, namely: (i) *eavesdrop* on communications, (ii) *intercept* traffic, (iii) *inject* any packet in the network (e.g., spoofed packets), and (iv) *flood* any network link they can access. Combining the listed ingredients, an attacker can perform complex attacks, such as man-in-the-middle (MITM) and DDoS attacks.

Although the attacker has virtually unlimited resources, we assume it is not capable of partitioning a network.

Assumption. *A network must guarantee the communication of a pair of entities if and only if there exists an adversary-free path between them*

Although this assumption might seem restricting, a network architecture cannot provide any availability guarantee if not met: an on-path attacker can trivially drop every packet, rendering communication impossible.

3.1.2 Device Security Capabilities

To systematically define what are the security capabilities that an IoT device must possess, we derive these capabilities from a document [NISft] released by the National Institute of Standards and Technologies (NIST). The document analyzes major frameworks on IoT security, from both industry and academia, and summarizes them in a list of device security capabilities. The ones that require network-level support are the following:

- device *availability*: an IoT device must be reachable by authorized entities at any moment in time,
- *access to interfaces*: an IoT device must restrict access to its network interfaces, and the protocols and services used by those interfaces, to authorized entities only,
- device *identification*: an IoT device must be uniquely identifiable, and
- *data protection*: an IoT device must protect the data it stores and transmits from unauthorized access and modification.

3.1.3 Network Security Services

We derive the security services a network must offer from the device security capabilities that require network-level support. Subsequently, we also refine these services to meet intra- and inter-domain specific features.

Guaranteed Access An authorized entity must be able, at any time, to communicate with a device. A network must guarantee access against both accidental failures and deliberate attacks. Practically speaking, the network must provide: (i) fast recovery from failures, and (ii) protection from performance degradation, via an access control system or a guaranteed bandwidth system, or both.

Restricted Access An authorized entity only must be able to communicate with a device, and the communication must use well-known interfaces only. Restricted access must be enforced both at the device and infrastructure level.

Authentication Communicating entities must authenticate themselves.

(🔒) Confidentiality Only the communicating entities must be able to obtain the content of a message. Moreover, since geographical compliance has recently become much required for privacy, we also require that the network offers *geo-fencing*, which is the ability to restrict the traffic inside a jurisdiction (e.g., a country or a banking consortium).

Table 3.1 shows the mapping between the security capabilities listed in Section 3.1.2 and the required network security services.

	Security Services			
				🔒
Device Availability	✓	✓	✗	✗
Device Identification	✗	✗	✓	✗
Data Protection	✗	✓	✓	✓
Access to Interfaces	✗	✓	✓	✗

Table 3.1: IoT devices security capabilities and the network services

Domain-Specific Features

For the remainder of this document, we use the word 'domain' to indicate a local-area network (LAN)-like environment, hence under the control of a single entity and characterized by private addressing and routing. Communication can then be of three types:

- *intra-domain*: happening inside a single domain only,
- *inter-domain*: going from the edge of a domain to the edge of another domain, possibly involving multiple hops, and
- *across-domain*: starting in one domain and ending in another domain, involving both intra- and inter-domain communication.

There are several use cases where an IoT system can span over multiple domains, thus involving across-domain communications. For example, multiple IoT deployments could be dislocated around the world and be under the control of one or more remote command-and-control (C&C) centers.

Each domain type interprets the network security services differently, adapting them to domain-specific characteristics. Moreover, each domain type requires different "operational features" that a network architecture must meet to be compatible with it.

Intra-Domain Security Services In an intra-domain setting, the security services listed in Section 3.1.3 translate to: (i) () device-to-device access must be guaranteed, (ii) () access to devices must be restricted, (iii) () identification of devices that communicate must be unique, and (iv) () device-to-device communication must remain confidential.

In a private environment, it is possible to know in advance the capacity that a network infrastructure must offer. A bandwidth reservation system is thus not required. Furthermore, if the network infrastructure is part of the Trusted Computing Base (TCB), confidentiality gets directly enforced by the network, and there is no need for end-to-end encryption.

Intra-Domain Operational Features The intra-domain network system must be "loose" () to adapt to the high dynamism of an IoT network, without making strong assumptions about the underlying network structure. Furthermore, it should be "light" () to be compatible with highly-constrained IoT devices.

Inter-Domain Security Features In an inter-domain setting, the security services translate to: (i) () domain-to-domain access must be guaranteed globally, (ii) () access to domains must be restricted, (iii) () the domains from which a communication originates must be identified, and (iv) () domain-to-domain communications must remain confidential.

Domain level authentication means that it is enough to identify from which domain a communication originated.

Inter-Domain Operational Features The inter-domain network solution must be "affordable" () from a financial and a resource allocation point of view – resources mean both time and people. Moreover, the solution should be "available" () to cover geographically distributed locations and must not be scarce.

Across-domain This case inherits the requirements of both the previous cases. When a message travels inside an intra-domain setting, intra-domain requirements apply, and when the message travels over an inter-domain setting, inter-domain requirements apply.

3.2 Intra-Domain Coverage

Notable past works on securing intra-domain networks are SANE [CGA+06] and its successor, Ethane [CFP+07]. Both systems provide fine-grained traffic control by using source routes as capabilities. Other works, such as DEADBOLT [KM18] and IoT Sentinel [MMH+17], instead, rely on the use of gateways to intercept traffic from nodes and perform decisions based on various rules (e.g., the current security state of the device). A more recent solution, which builds upon previous capability-based and deny-by-default systems, is LaNeCa (see Section 2.1).

3.2.1 Solutions comparison

	Security Services				Operational Features	
SANE	✓	✓	✓	✓[1]	✗	✗
ETHANE	✓	✓	✓	✓[1]	✗	✗
DEADBOLT	✓	✓	✓	✓[1]	✗	✗
IoT Sentinel	✓	✓	✓	✓[1]	✗	✗
LaNeCa	✓	✓	✓	✓[1]	✓	✓

Table 3.2: Intra-domain systems comparison

SANE and ETHANE SANE and ETHANE meet all the required security features: they guarantee access by early dropping unauthorized traffic, they make it possible to allow only authorized flow to reach devices, and they all provide per-device authentication means. However, they both pose strong assumptions on the structure of the network by using source routes as capabilities. Moreover, although switches in ETHANE and SANE require modest resources, and with some modifications, they could likely be run on IoT devices as well, the required constant connection with the controller is itself extremely demanding for IoT devices.

DEADBOLT and IoT Sentinel Like SANE and ETHANE, DEADBOLT and IoT Sentinel meet all the required security features in the same way. However, they both require the traffic involving an IoT device to be mediated by an access point (AP). They then both pose strong assumptions on the network and fail at being light systems since an AP just cannot run on an IoT device.

LaNeCa Unlike the other systems, LaNeCa covers both the required security and operational features. Security features are achieved similarly to the other systems. The traffic control mechanism, based on local-only knowledge and symmetric cryptography, goes well with IoT devices, as highlighted by the authors when highlighting that for an IoT device, transceiving a packet requires multiple orders of magnitude more energy than performing (symmetric) cryptography [HH18, SHNN12]. Moreover, traffic gets controlled by every node in the network, which makes it deployable in mesh-up networks as well. For these reasons, we employ LaNeCa to handle the intra-domain traffic of our system.

[1] Confidentiality is not directly offered by the system but can be built on top of its authentication system

3.3 Inter-Domain Coverage

Similarly to the intra-domain, we inspect various options that could be employed to handle the inter-domain communication of our system.

The Internet is the obvious solution when it comes to interconnecting networks worldwide, being cost-effective and pervasive. Another option is SCION, which also uses public infrastructure, but addresses shortcomings of the current Internet. Finally, dedicated private connections, which ensure physical isolation from other users, can be used to interconnect remote networks.

3.3.1 Solutions comparison

	Security Services				Operational Features	
Private	✓	✓	✓	✓	✗	✗
Internet	✗	✗	✓	✓[2]	✓	✓
SCION	✓	✓	✓	✓[2]	✓	✓

Table 3.3: Inter-domain solutions comparison

Private

Private solutions, like Multiprotocol Label Switching (MPLS), can offer on-demand Quality of Service (QoS) both in terms of performances and availability guarantees. Moreover, confidentiality and authentication are achieved with physical (or virtual) isolation. However, they come at a very high cost as well, being particularly demanding both in terms of monetary and management effort, thus failing at being affordable. This lack of agility is further exacerbated when multiple internet service providers (ISPs) and Internet exchange points (IXPs) are involved in a message transit – which happens to be very likely when covering global distances.

It is no surprise that companies are moving away from private solutions in favor of fully public or hybrid (software-defined networking in a Wide Area Network (SD-WAN)) ones [Gar20c].

Internet

The Internet is a network of networks, composed of several thousands of ASes scattered all over the world.

[2] Confidentiality is not directly offered by the system but can be built on top of its authentication system

BGP These ASes are glued together with a protocol called Border Gateway Protocol (BGP), being the de facto standard exterior gateway protocol of the Internet.

Yet, BGP fails at securing the Internet's worldwide routing against both deliberate attacks [Wik20b], with the most common being BGP hijacking [Clo20c], and accidental misconfigurations [Clo20a]. It is no secret that BGP presents security deficiencies that led to thousands of incidents every year [Int17]. Attempts have been made to render BGP more secure but were not successful [CHB+13, SVG13]. When dealing with such complex systems, it is not always possible to adapt them to target new use cases, and BGP is the proof for this proposition. Furthermore, new cracks can show up while trying to fix current ones [LHZ14], merely moving the weaknesses around.

Failure Recovery When a connection between ASes fails, BGP is to react; however, it does not shine for reactivity, since getting the Internet back to a stable state may take up to tens of minutes [LABJ01, OZPZ09] – this is true even during physiological routing updates. This lack of agility to adapt to changes poses a significant barrier for the adoption of IP-based solutions to replace ad hoc communication means (e.g., VoIP [KKK07]). Although failure might seem and edge case, when considering a large number of nodes and links, it is likely to happen.

Guaranteed Bandwidth Several QoS mechanisms and schemes exist for Internet Protocol (IP) networking but the public Internet does not implement any of them. To still provide a high-quality connection to their customer, ISPs over-provision their infrastructure to be able to cope with peak traffic loads estimates. While this solution is acceptable for some applications, it is still a best-effort approach, and does not provide any real guarantee.

DDoS Protection The Internet does not offer network-level access control. However, a similar result is achievable with the combination of *Anycast* [Wik20a] routing and edge computing [Wik20d]. The combination of these two paradigms allows moving traffic access control closer to its source, so to intercept traffic before it can reach its destination and impact the network infrastructure. Although Anycast is not standardized in Internet Protocol version 4 (IPv4) – the de facto standard protocol used by the Internet, several companies such as Cloudflare, Akamai, Amazon, Microsoft, and Google offer solutions against DDoS attacks[3]. These services all share a common defense strategy: each company owns several points of presence (PoPs) in strategic places all over the world that function as edge nodes. Each PoP offers two services:

1. *Content Delivery Network (CDN)*: to serve content stored at the edge, so to offload the origin server, and

[3]We only consider pure network-level attacks only, excluding L4 (such as TCP SYN flood [Wik20e]) or L7 attacks (such as HTTP flood attacks [Clo20b])

2. *reverse proxying*: to perform traffic filtering, employing heuristics that span from simple IP blacklisting to Deep packet inspection (DPI), and load balancing.

Nonetheless, to offer such services, Anycast routing must still be achieved. A workaround used by companies to emulate Anycast consists of using BGP to announce IP addresses, owned by their customers, at each PoP. Then, every AS automagically routes the traffic directed to their customers via the closest PoP, giving providers the chance to intercept traffic as early as possible and reroute it via their private premises. While the solution is not flawed itself, it uses BGP, which, as we have previously discussed, is a risk itself.

SCION

The SCION architecture uses public infrastructure and, as such, inherits all the advantages of the Internet. Yet, it was built from the ground up to address most of the Internet shortcomings by design, starting with being BGP free.

Failure Recovery The multi-path nature of SCION makes it extremely resilient to link failures: circumventing a failure is just a matter of routing the traffic over another path, which can be carried out directly by the sender. Moreover, the routing architecture does not need to convergence to any stable state, so failures do not impact the routing control plane.

Guaranteed Bandwidth There is an ongoing effort to achieve a global QoS with the Scalable Internet Bandwidth Reservation Architecture (SIBRA) [BRS+15] system. However, the system is still under development and not implementable in the scope of this thesis – we still plan to integrate SIBRA into our work in the near future.

DDoS protection A network-level access control, as a DDoS defense mechanism, can be built on top of the SCION pathing system. This system is called *hidden paths*, and the rationale behind it is that only a group of authorized ASes can access certain links, blocking malicious traffic at the network level. The result is similar to what Internet DDoS prevention systems offer but with two main differences: (i) no need to use BGP at all, and (ii) traffic filtering is backed by cryptography, instead of heuristic matching. Last but not least, by offering a scalable, efficient, and secure point-to-point communication means, SCION can be the starting point for more complex communication paradigms such as Anycast, laying the foundation for higher-level services such as CDNs.

All in all, SCION offers strong guarantees combined with low entry barriers, comparable to that of the Internet, and provides all the security features needed for an inter-domain protocol compatible with our system.

4 Implementing LaNeCa

This chapter explores the L2.5 implementation of LaNeCa. First, we analyze how it integrates with the popular lightweight IP (lwIP) network stack. Then, we focus on building the cryptographic primitives used for key sets derivation and tag checking. Finally, we present the LaNeCa controller. Please note that most of the implementation is independent of the level at which LaNeCa operates. Different LaNeCa implementations (e.g., L3.5 LaNeCa) would only need to adapt the ingress and egress shims, and other minor details.

4.1 Integrating LaNeCa in lwIP

To make LaNeCa the most compatible with existing IoT applications, we extend the widely adopted, open-source lwIP protocol stack [Dun01]. Two characteristics of lwIP made it the default choice for several IoT solutions: (i) as the name suggests, it is remarkably lightweight, both memory and a computationally wise, and (ii) it can seamlessly run with or without an underlying Operating System (OS) such as FreeRTOS. These characteristics make it a perfect fit for constrained devices. To achieve maximum reproducibility, we do not use an OS, but rather, we run lwIP in 'bare metal' mode. Still, the implementation is fully compatible with an OS too.

4.1.1 An L2.5 Protocol

LaNeCa integrates with lwIP stack similarly to virtual LAN (VLAN) [IEE20a]. Like VLAN, LaNeCa is an L2.5 protocol, sitting in between L2 (data link layer) and L3 (network layer). Practically speaking, this means that it is considered an L3 protocol by Ethernet and is invisible to L3 protocols. Being an L3 protocol for Ethernet, we need to allocate an *EtherType* specifically for it: we use 0x83CA.

Mapping LaNeCa entities to L2 attributes In this implementation of LaNeCa, a flow is identified by the medium access control (MAC) addresses of the sender and the receiver, and by the L3 protocol used to communicate.

The LaNeCa Header

Fig. 4.1 shows the structure of a LaNeCa header. In this example, a Tag is 48 bits long and is composed of 3, 16-bits MACs.

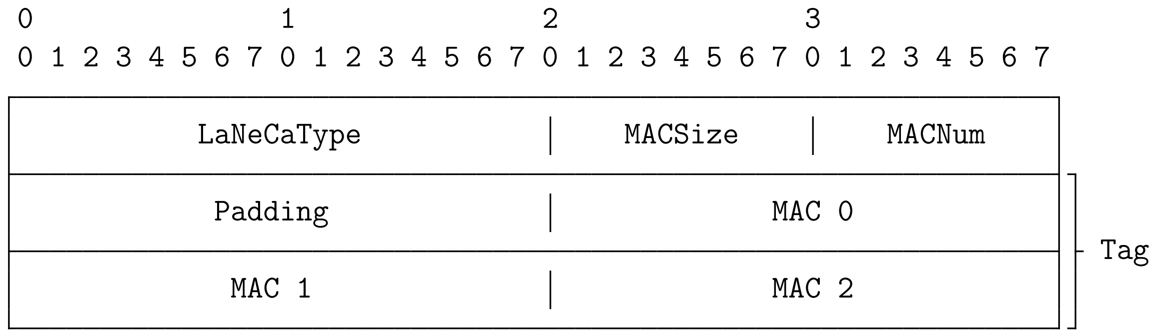


Figure 4.1: The LaNeCa header structure

LaNeCaType To indicate the protocol of a flow, we cannot use the protocol field in the header of an Ethernet packet. Instead, the protocol is placed in the *LaNeCaType*. For full compatibility with Ethernet's *EtherType*, the field is two bytes long.

MACSize and MACNum A LaNeCa tag is composed of MACs, which can vary in size and number. How big a MAC is can be found in the *MACSize* field, while the number of MACs composing the tag is located in the *MACNum* field.

On the size of the fields Even if both fields are 8 bits (1 Byte) long, it does not make sense them in their entirety, since it would mean having a tag size of $(2^8 \times 2^8) - 1 = 65\,535$ Bytes. Although a tag length of 256 Bytes is enough for any application, using only a byte to include both the size and the number of MACs would be limiting for some applications (cfr. Sections 4.4.1 and 4.4.2). For the MACNum field, we would need at least 4 bits to account for use cases that require up to 16 MACs, considering that we can use the MACNum field with a plus one offset since there has to be at least a MAC in the packet. Such an allocation leaves us with 4 bits for the MACSize, which means that the biggest MAC can be 16 Bytes, which is not enough for use cases that require a single, long MAC.

Padding A variable-sized buffer with zeros is placed before the tag to memory-align a LaNeCa payload. Its usage is described in Section 4.2.1.

Macs (Tag) The Tag field contains the raw list of MACs of a packet.

Adjusting Pbufs Allocations

In lwIP, packets are built from a sequence (linked-list) of one or more data structures called packet buffer (PBUF). The most important fields of a PBUF are: (i) a memory buffer containing the data of the packet, (ii) a pointer (`payload`) to an offset in the memory buffer, and (iii) some metadata for PBUF handling operations (such as the pointer to the next PBUF).

PBUF Layers PBUF layers are used to pre-allocate a buffer big enough to hold the payload and all the headers of a packet. Such a pre-allocation avoids copies and further allocations to attach headers down the network stack. The PBUF layers, in order, are `PBUF_TRANSPORT`, `PBUF_IP`, `PBUF_LINK`, `PBUF_RAW_TX`, and `PBUF_RAW`. Each layer, from left to right, includes the size of the following layer, recursively including all the following headers. For instance, to send a message of size n over UDP, a buffer of size PBUF_TRANSPORT + n gets allocated. The `p->payload` pointer would then point to n bytes from its end, making the process of prepending other headers just a matter of moving the `p->payload` pointer back.

Integrate LaNeCa in Layers Since we are transparently stacking an additional header in the PBUF, we need to transparently change the size of PBUF allocations to make room for a LaNeCa header. We do this by adjusting the size of the `PBUF_LINK` layer size. This adjustment propagates to all PBUF allocations of upper layers, without requiring any changes.

4.1.2 Ingress Shim

To keep the lwIP API untouched, we try to leverage some already defined hook to add a new EtherType handler. Fortunately, lwIP allows us to define a custom hook to intercept unknown Ethernet protocols by defining the optional macro `LWIP_HOOK_UNKNOWN_ETH_PROTOCOL`. As shown in Fig. 4.2, we can add our LaNeCa packet handler in the ingress stack without touching it. Wiring the LaNeCa input

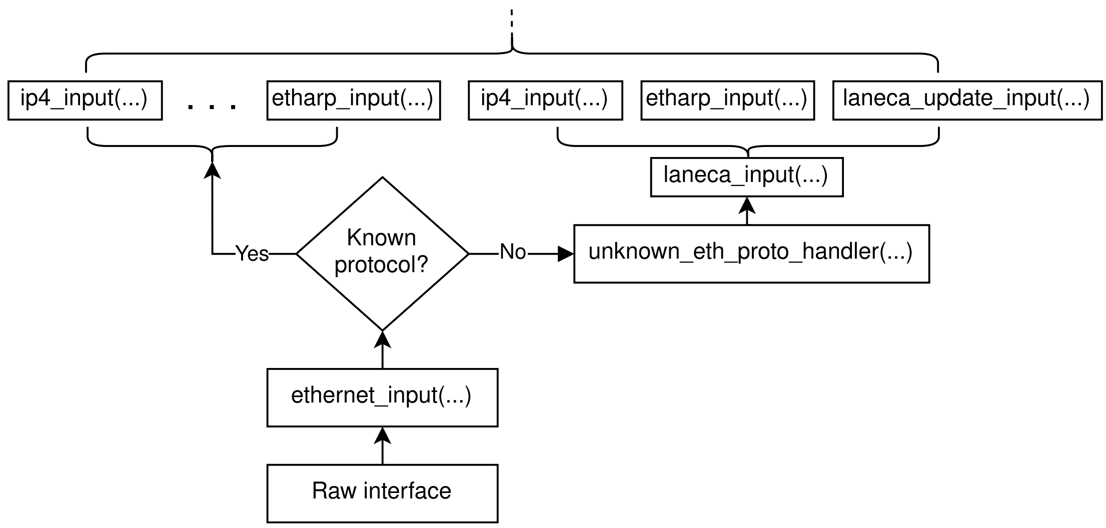


Figure 4.2: Hook to intercept received LaNeCa packets

function back to the `ethernet_input(...)` switch would have been cleaner, but the latter expects a PBUF with an Ethernet payload inside as input, instead of just a Protocol Data Unit (PDU) and the EtherType of the PDU. Reassembling an Ethernet PDU in a PBUF is not worth the code saving.

LaNeCa Input Function

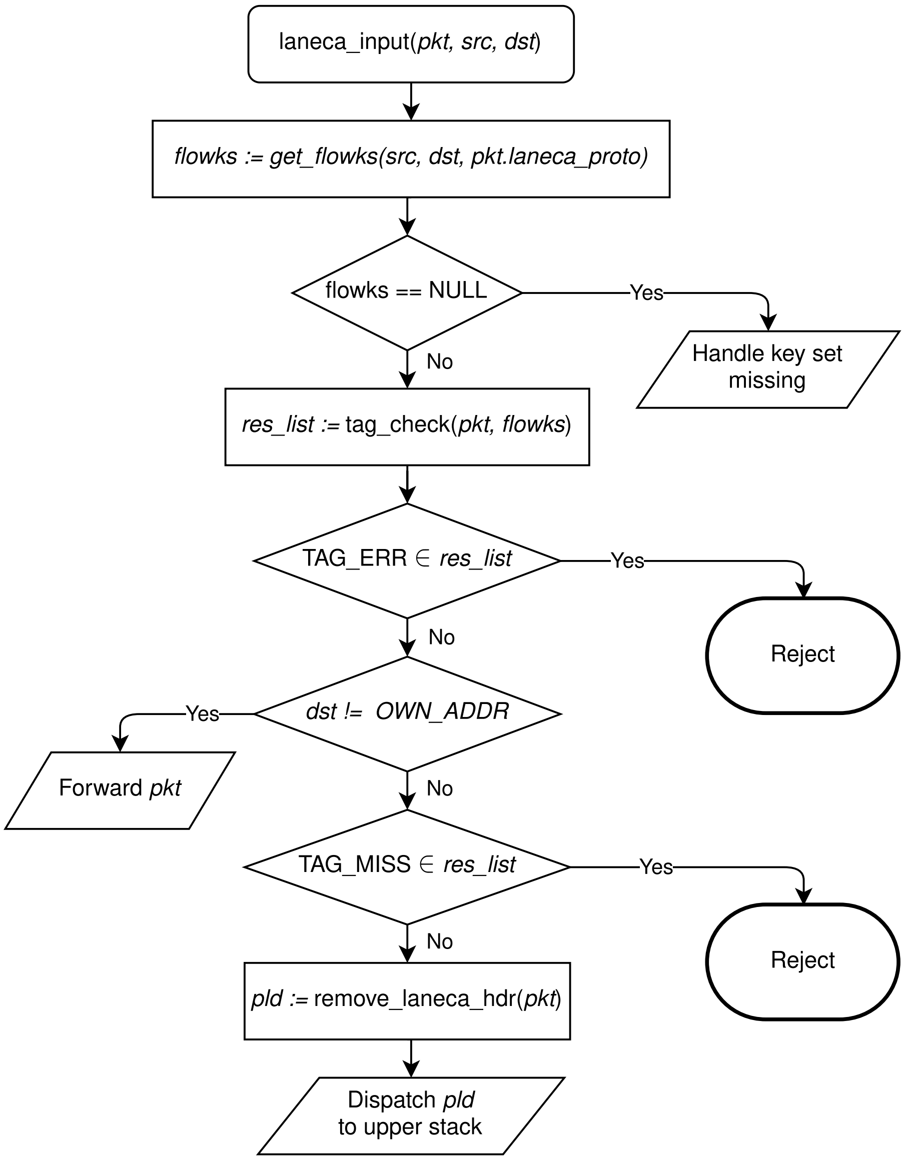


Figure 4.3: LaNeCa input packets handler

The `laneca_input(...)` function (Fig. 4.3) checks the authorization of every incoming packet. The input handler performs three operations: (i) retrieves a flow key set compatible with packet's flow, (ii) checks the legitimacy of the packet, and (iii) either forwards the packet to other devices or up its network stack. If `get_flowks(...)`, whose pseudo-code is listed in Fig. 4.4, does not return any key set, the behavior of the node depends on the environment's configuration. If dynamic key management via the LaNeCa controller (more in Section 4.3) is enabled, the node initiates the key sets retrieval procedure. Otherwise, the packet gets just dropped. If instead, dynamic key management is not active, such a miss might be

```
def get_flowks(laneca_conf, src, dst, proto):
  for flowks in laneca_conf.flow_key_sets:
    if flowks.is_flow_for(src, dst, proto):
      return flowks
  for recvks in laneca_conf.recv_key_sets:
    if recvks.is_recv_for(dst):
      return get_flowks_from_recvks(recvks, src, proto)
  if laneca_conf.has_rootks:
    recvks = get_recvks_from_rootks(dst)
    flowks = get_flowks_from_recvks(recvks, src, proto)
    return flowks
  return NULL
```

Figure 4.4: Pseudo-code of the `get_flowks` function

a symptom of a network misconfiguration. Once a flow key set is retrieved, the node verifies the tag of the packet with the tag check function (depicted in Fig. 4.5). Depending on the context, the caller can perform further checks on the results. For instance, a receiver cannot accept that a MAC check gets skipped. In this case, a `TAG_MISS` in the `res_list` shall be considered an error.

4.1.3 Egress Shim

Similarly to the ingress shim, it would be ideal to leverage some hook to add a handler in the output stack before passing a PBUF to the L2 output handler. Unfortunately, and reasonably, lwIP does not offer any similar hook. Recalling that LaNeCa is an L2.5 protocol, the output handler needs to intercept calls to the Ethernet output handler, so that it can add a LaNeCa header to the PBUF, and then call the Ethernet output function. To keep the lwIP application programming interface (API) untouched, we rename the original Ethernet output handler (from `ethernet_output(...)` to `_ethernet_output(...)`), and we replace it with a relay function that simply calls `laneca_output(...)`, as shown in Fig. 4.6. Another option would have been changing the initialization function `ethernetif_init(...)` of the network interface, where the pointer to its output function (`netif->output`) is specified. However, by default, it points to `etharp_output(...)` that, as the name suggests, implements Address Resolution Protocol (ARP) and is called by the IP layer to output packets. It is `etharp_output(...)` that then in turn calls `ethernet_output(...)`. Hence, by performing the rewiring as in Fig. 4.6, we support ARP as well.

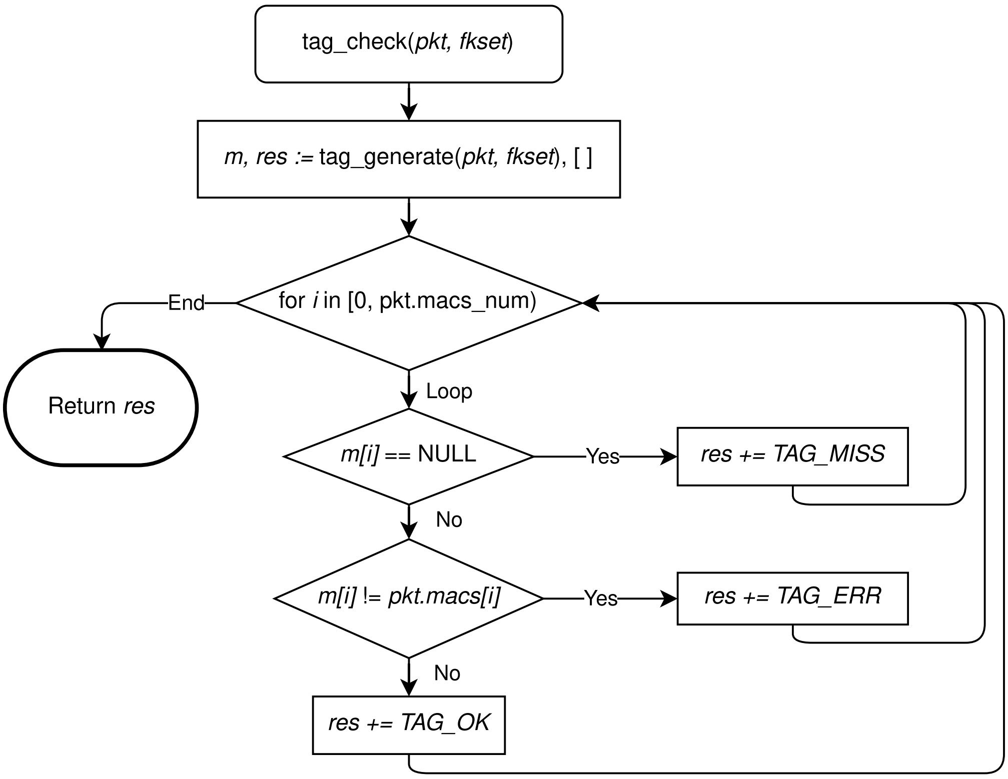


Figure 4.5: LaNeCa tag check function

LaNeCa Output Function

The `laneca_output(...)` function (Fig. 4.7) encapsulates an L3 PDU into a LaNeCa packet. The output handler performs three operations: (i) retrieves a flow key set compatible with the packet's flow, (ii) adds a LaNeCa header to the packet, and (iii) forwards the packet down the stack. Likewise to the input function, if `get_flowks(...)` does not return any key set, the device might either ask the controller for a key set or report a network misconfiguration.

4.2 Handling LaNeCa Cryptography

Cryptography is at the heart of LaNeCa. In this chapter, we explain how we realize its cryptographic toolbox to verify tags and derive key sets.

4.2.1 Cryptographic Primitives

In this section, we cover the implementation of the cryptographic primitives used by higher level functions (e.g., PRFs).

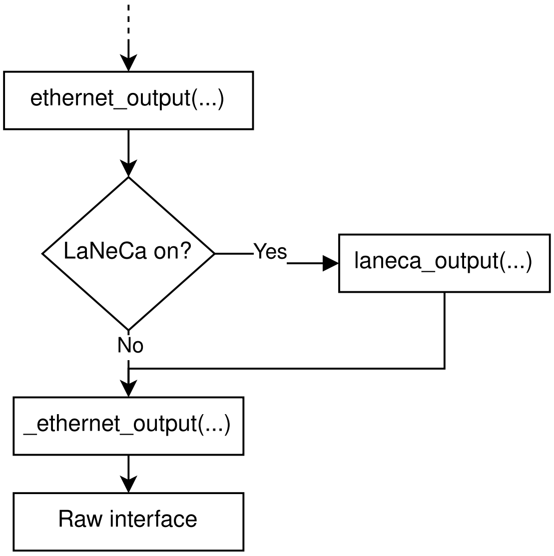


Figure 4.6: lwIP output stack rewiring to add LaNeCa output handler

DMA Mode and LaNeCa Header Padding Field

Hardware-accelerated crypto modules are critical when performing cryptographic operations at line-rate. These modules are usually separated from the microcontroller unit (MCU) and cannot access data stored in RAM. Moving data back and forth between random-access memory (RAM) and modules registers might be expensive. Fortunately, the overhead of this operation can be dramatically reduced using a component called Direct Memory Access (DMA) controller. The DMA controller, itself external, takes care of rapidly moving data from memory to memory and from memory to peripherals (and vice versa). One requirement for using DMA is that the source and destination memory addresses must be physically aligned. Fulfilling this requirement when working with memory under our control is not a problem. However, in our case, we inherit the memory from the network stack.

A naive solution might be to change the allocation of an entire packet to make the Service Data Unit (SDU) of LaNeCa aligned. However, the memory used to store a whole packet must be aligned too, since the data received by the Network Interface Card (NIC) is loaded in memory via DMA as well. To solve this problem, we introduce a padding field that ensures the LaNeCa SDU is already aligned when received from the network.

Calculating the Padding The padding p for an alignment a is calculated as in Eq. (4.1), where $ehdr$ is the size of Ethernet header (fixed at 14 bytes) and $nhdr_{nopad}$ is the size of the LaNeCa header without padding.

$$p = \begin{cases} 0 & \text{if } (ehdr + nhdr_{nopad} \bmod a) = 0 \\ a - r & \text{otherwise} \end{cases} \tag{4.1}$$

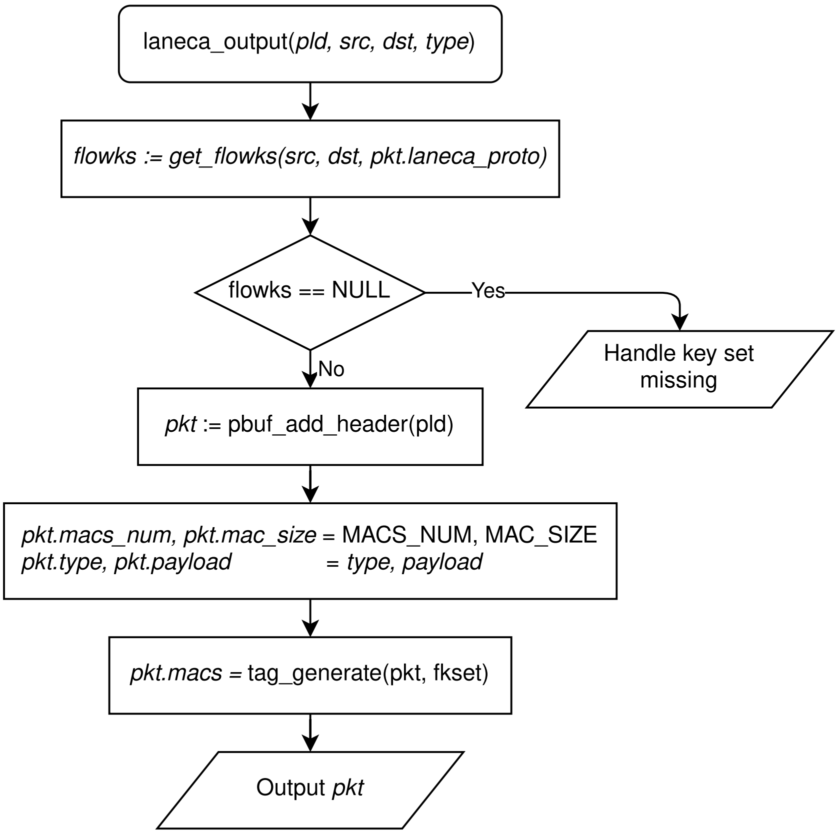


Figure 4.7: LaNeCa output function

MAC Functions Alternatives

In this section, we present some alternatives for computing message authentication codes (MACs).

CBC-MAC We can realize an efficient MAC computation function with a cipher block chaining message authentication code (CBC-MAC) [Wik19]. This technique consists of constructing a MAC from a block cipher. We can then use Advanced Encryption Standard (AES)-Cipher Block Chaining (CBC) (AES-CBC) as the block cipher of CBC-MAC. In short, the method consists of performing an AES encryption of the message, using an initialization vector of zeros, and considering only the last block. Thanks to the chaining nature of AES-CBC, the last block of the computation depends on the entire message and can be used to authenticate it.

However, when dealing with messages of variable length, CBC-MAC presents serious security deficiencies [Gre13]. Such deficiencies allow an attacker to forge valid tags for arbitrary messages, hindering the security of the whole system.

CMAC A flavor of CBC-MAC that accounts for its flaws is Cipher-based Message Authentication Code (CMAC) [SPLI06], suggested by the NIST [NIS05] as a drop-in replacement for CBC-MAC. We can adapt the C implementation provided by RFC4493 to make it compatible with the hardware crypto module of the boards. However, while the software implementation of CMAC and CBC-MAC is almost computationally identical, things change when using a hardware acceleration. Unlike CBC-MAC, CMAC treats the last AES block differently, depending on the size of the payload, making it incompatible with default AES-CBC chaining, and requiring more than one go of the crypto module.

CBC-MAC with pre-hashing It might then be profitable to still use CBC-MAC by taking some precautions. We can reduce the input of CBC-MAC to a fixed length by pre-hashing the message. From now on, we refer to the CMAC mode of operation with Plain mode and the CBC-MAC with pre-hashing mode of operation with Pre-hashing mode. The tag generation process is depicted in Fig. 4.8. It can operate in two modes, a Plain and a Pre-hash mode.

We perform an in-depth analysis of the two modes later in Section 6.2.2. Although it is common to find hardware-accelerated crypto modules in most IoT devices, we also provide a software only implementation.

PRF for Key Derivation

Key derivation in LaNeCa is based on a PRF (Section 2.1.1). To implement such a function we leverage once more CBC-MAC, which can be safely used since the input is always of the same size. The key derivation starts from a root key set $\mathcal{K}$ of n keys. A receiver key set for a receiver r is derived with the following:

$$\mathcal{K}^r = \{k_i^r = \text{CBC-MAC}_{k_i}(r_{\text{MAC}}) \mid i \in \{1, \ldots, n\}\} \tag{4.2}$$

where

CBC-MAC_{k_i} = CBC-MAC keyed with root key k_i
r_{MAC} = byte array of the MAC of receiver r

A flow key set for a sender s with protocol p is instead derived with the following:

$$\mathcal{K}^{r \leftarrow s:p} = \{k_i^{r \leftarrow s:p} = \text{CBC-MAC}_{k_i^r}(s_{\text{MAC}} \| p_{\text{Type}}) \mid i \in \{1, \ldots, n\}\} \tag{4.3}$$

where

$\text{CBC-MAC}_{k_i^r}$ = CBC-MAC keyed with receiver key k_i^r
s_{MAC} = byte array of the MAC of sender s
p_{Type} = is the EtherType of the protocol used to communicate with r

4.2.2 Key Sets Handling

Each device in a LaNeCa network must be able to handle key sets. To capture their hierarchical relation, we need a lightweight tree-like data structure to hold

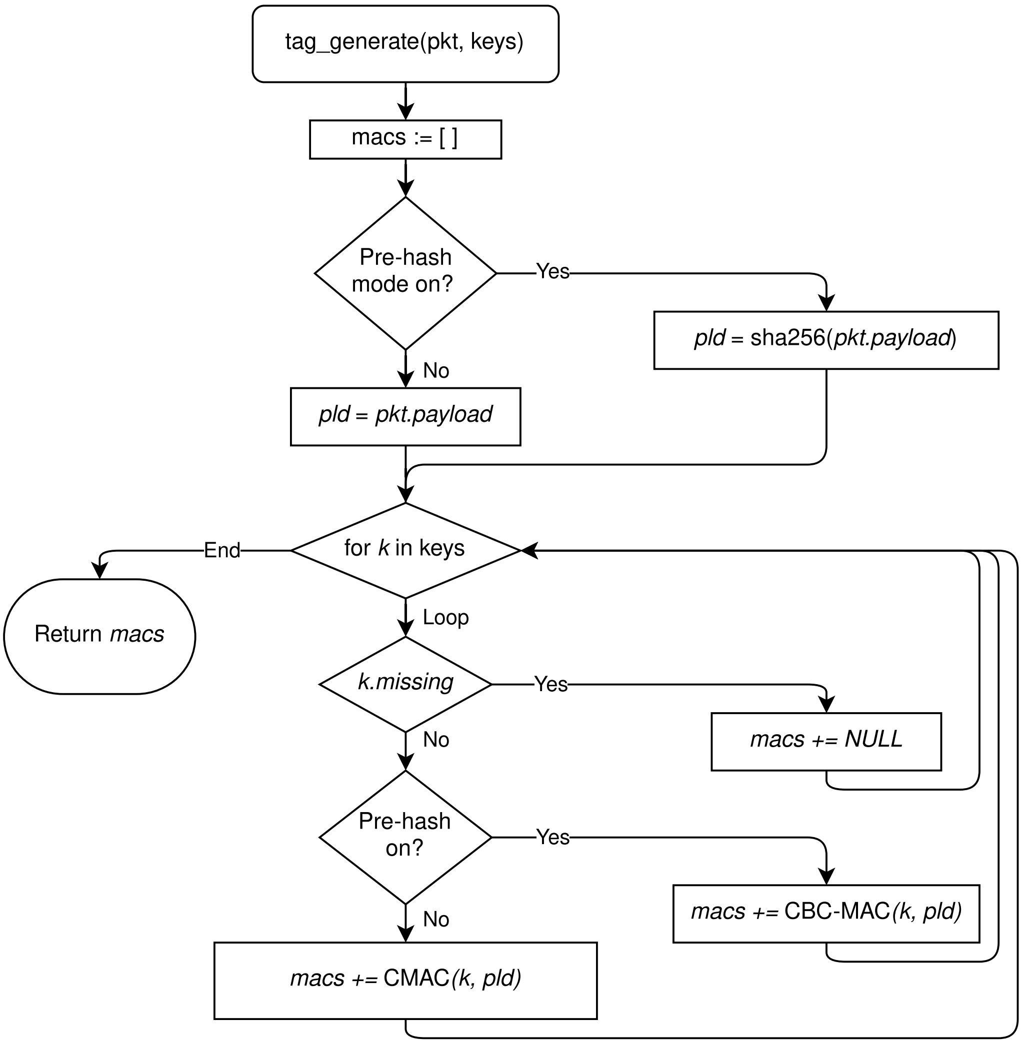


Figure 4.8: Tag generation process

them, taking into account that, depending on the device functions, we might end up using only a fraction of the type of keys sets (e.g., a device that only pushes sensor readings to a server will never need root or receiver level keys). Figure 4.9 shows the structures involved in the management of a LaNeCa configuration.

laneca_conf_t An instance of `laneca_conf_t` captures the root of a LaNeCa configuration. The `keys_handlers` field is a *deque* [Wik20c] containing instances of `keys_handler_node_t`.

keys_handler_t A `keys_handler_t` is the shared basis of any key set. Since certain key sets operations are the same for all the types, we realize a simple inheritance mechanism in C. An explicit type-hinting, needed to discern among types, is located in the `level` field.

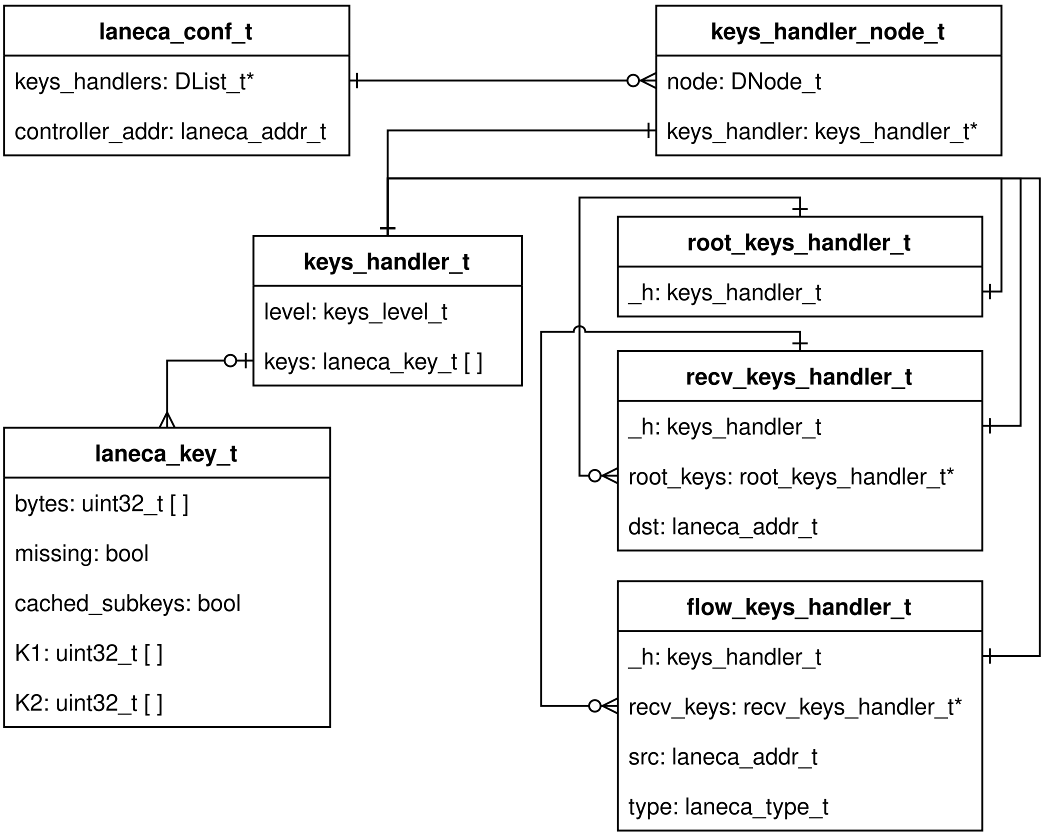


Figure 4.9: Key sets handlers

laneca_key_t A `laneca_key_t` ultimately holds a LaNeCa key. Also, this structure retains the cached keys of a CMAC computation and indicates whether the held key is missing or not. On a side note, it would be correct, in the implementation specification, to call a "key set" a "key list". However, for the sake of consistency with the LaNeCa protocol, we keep calling "key sets" as such but bear in mind that they are lists.

4.3 LaNeCa Controller

A LaNeCa controller manages the dispatching of key sets in a LaNeCa deployment. The advantages of employing a controller are the following:

1. setting up a network requires minimum effort: with the controller key sets exchange protocol, devices can ask for key sets on-demand,

2. traffic rules can be added along the way: if an additional flow is authorized, devices that intend to communicate on that flow will receive required key once requested, and

3. new nodes can be easily added to the network: adding new nodes is just a matter of updating the controller's policies (see Section 4.3.2 for more info).

If we can think of switches and end-devices as the muscles of LaNeCa, the controller is its brain. Although the controller carries out special duties and is the only node in the network that owns the full root key set, from a traffic control perspective it is still a regular LaNeCa node. Hence, to send a receive messages, they must be authenticated LaNeCa packets. Furthermore, like every other node, if enabled, the controller also acts as a switch.

4.3.1 Bootstrapping the Control Plane

When setting up a network using a controller, we need to send to the nodes the key sets they need to operate. When exchanging a key set, be it root, receiver, or flow type, we must take some precautions, and the following guarantees are needed:

1. both the node that requests a key set and the controller must be authenticated, and

2. if the network infrastructure is outside the TCB, given that the exchanged messages contain secrets, the communication must be confidential.

To construct an authenticated channel, we need a pre-shared knowledge between the parties. A solution might be to generate and distribute a random symmetric secret to each (controller, device) pair, or use asymmetric cryptography to authenticate the parties. However, this would lead to the introduction of exceptions to traffic control at bootstrap time, which would hinder the deny-by-default nature of LaNeCa. To circumvent this issue, we need a way to self bootstrap LaNeCa.

Observation. *At any moment in time, only three entities can generate a valid tag*[1] *for the flow identified by the triple (sender, protocol, receiver):*

1. *the sender, which owns the flow key for the said triplet,*

2. *the receiver, which can derive flow key sets for messages targeted at itself starting from its receiver key set, and*

3. *the controller, which can derive every key set starting from the root keys that are all known to it.*

In this case, however, the controller *is* the receiver. Thus, if we provide a node with a flow key set to communicate with the controller, they can uniquely authenticate each other.

As far as confidentiality is concerned, nodes can derive an encryption key in multiple ways, to name a few:

[1] under the assumption that the LaNeCa infrastructure has not been compromised

1. they can combine the keys in a flow keys set to derive a new key,
2. they can sequentially encrypt the exchanged messages using all the keys in the flow key set, or
3. they can execute a key exchange algorithm (such as DH) to agree on an encryption key.

Although the first two solutions are not efficient, they offer a 0-RTT key agreement. Combined with the fact that control plane messages are rarely exchanged, they might be valid options in certain situations.

4.3.2 Policies

Policies are a fine-grained and concise way for administrators to describe a LaNeCa configuration, to instruct the controller on how to dispatch keys.[2]. There are two types of policies: (i) *root*: to configure the distribution of root keys, and (ii) *flow*: to describe which flows are allowed. The root policy specifies the mechanism used to assign root keys to a security equivalence class, in this case, based on likelihood. Moreover, it also specifies how the security equivalence classes are composed. Unlike the root policy, flow policies can be multiple. A flow policy is a rule that allows a group of devices $\mathcal{A}$ to communicate with a group of devices $\mathcal{B}$ over a group of protocols $\mathcal{P}$, with a specific directionality, namely unidirectional (client/server) or bidirectional (peer).

```
root:
  prob: 0.4
  classes: [['AA:AA', 'BB:BB'],['55:55', 'CC:CC', 'DD:DD']]
flow:
  - {mode: 'client/server', clients: ['11:11'], servers: ['33:33']
   ↪  protocols: ['IP']}
  - {mode: 'peer', peers: ['44:44', '55:55'], protocols: ['IP']}
```

Figure 4.10: An example of a policy configuration

Figure 4.10 shows an example of a policy configuration. In this scenario, each security equivalence class will have, in expectation, 40% of the root keys. The first flow policy allows node `11:11` to send messages to node `33:33` over IP. The second flow policy permits a free exchange of messages among nodes `44:44` and `55:55` over IP.

[2]Root keys distribution has not been implemented yet

4.4 Use Cases

In this chapter, we present two use cases very different from each other, showcasing the unique adaptability of LaNeCa.

4.4.1 Smart Factory

Consider a smart factory that leverages an IoT mesh-up network to control its production processes. The network of the factory can easily contain ∼500 devices, ranging from full-fledged automated robots to ultra-low-powered air quality sensors. Still, the number of device types would likely be around 10. Moreover, we can assume that the network provides wireless gateways situated in ∼5 environments, all with different security measures. We then have a network with 15 security equivalence classes: (i) one for each device type, and (ii) one for each environment. We configure LaNeCa with a root key set of size $n = 10$ and assign each root key to a security equivalence class with a 30% chance – devices in the same security equivalence class receive the same root keys. The Mac Size is set to 2 Bytes (16 bits), and each tag will then be 160 bits = 20 bytes long, more than the recommended minimum size of 128-bit [BBB+06, OH04]. In expectation, 4 MACs are verified at each hop. A security analysis of the most promising attacks that could be performed in this scenario is provided in Section 6.1.

4.4.2 Smart Home

A home network is usually characterized by several devices connected to a single AP. Since no device, except for the AP, forwards packets, there is a single security equivalence class, and a single root key is sufficient. Moreover, the AP can also act as the controller of the deployment. To obtain a tag of a recommended size we choose a Mac Size of 16 Bytes (128 bits). While using only a root key might result in a trivial scheme, this shows how LaNeCa can adapt to various settings, while still drastically improving the overall security.

The only way to compromise the system is by compromising the AP, since trying to brute-force the 128 bits tag, to overcome even only the first-hop checks, is not feasible.

5 SENECA: SCION augmEnted LaNECA

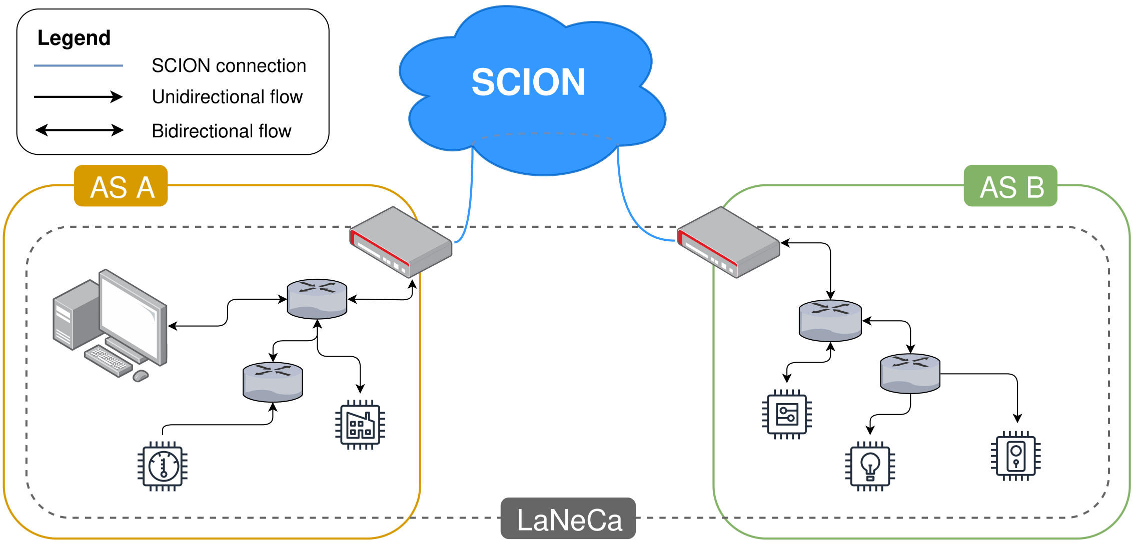


Figure 5.1: SCION augmEnted LaNECA (SENECA)

This chapter explains how we employ SCION to scale LaNeCa globally. First, we explore how we use SCION to relay LaNeCa traffic across domains in a transparent wa in a transparent wayy. Second, we explain how we achieve E2E encryption between gateways, hidden paths, and fast failover. Finally, we show how we leverage the relaying mechanism used for LaNeCa traffic as a tunnel for legacy IP applications.

5.1 LaNeCa Traffic Relaying

We extend the reach of LaNeCa by transparently interconnecting different LaNeCa domains with SCION. At the edge of each LaNeCa domain, we place gateways that exchange messages directed at devices in foreign domains. The overall idea is depicted in Fig. 5.1. Each gateway exposes two ports: a control plane and a data plane port. Separating control from data traffic makes it possible to achieve a complex control plane and keep the data plane lean and fast. Since we relay intra-domain traffic over another protocol, similar to a Virtual Private Network (VPN) tunnel, we employ

UDP/SCION as the communication protocol between the gateways. The reasons for choosing User Datagram Protocol (UDP) are:

1. UDP tunnels have proven to provide lower latency than Transmission Control Protocol (TCP) tunnels [CCS15], and

2. TCP over TCP might lead to a problem called 'TCP meltdown': in short, the 'reliability' mechanisms of the stacked protocols interfere with each other leading to a degraded connection.

Fig. A.1 shows an example configuration for the gateway placed in as AS A. At the beginning of the configuration, we specify the gateway's own SCION address, which is needed to register itself at the local *scion dispatcher*, along with the control and data ports. For each remote gateway, we specify its SCION address, their control and data ports, and an optional rendezvous AS to establish a hidden path (more in Section 5.3).

Each gateway also acts as a LaNeCa controller: the `laneca` section of the configuration specifies which devices are in the gateway's domain, and also what are the domain's policies. Across-domain flow policies get synced among the gateways that hold the various devices. Finally, to handle LaNeCa traffic, we need to listen for incoming L2 packets, and this requires listening on a specific `interface`. Moreover, we use the same interface to dispatch incoming remote traffic[1].

The "pathing" and "ip" options are covered in Sections 5.4 and 5.5 respectively.

5.2 Gateways for E2E Encryption

End-to-end encryption between gateways serves two purposes:

1. as required by Section 3.1.3, inter-domain traffic shall be domain-to-domain encrypted, and

2. a confidential channel is needed to bootstrap hidden paths (more in Section 5.3).

The first step to establish a secure channel between each pair of gateways is to establish an authenticated channel. Authentication is critical to stop a MITM attack from completely vanishing the confidentiality efforts, which, as we specified in Section 3.1.1, can be carried out by the attacker. To authenticate a gateway, we leverage the Pragmatic Internet-Scale Key-Establishment System (PISKES) [RRLP20], which offers a fast way to obtain a symmetric key to sign messages efficiently. Furthermore, being integrated into SCION and being bootstrapped with the SCION control-plane public-key-infrastructure (CP-PKI), it is possible to leverage PISKES to authenticate ASes. Since we consider the AS that hosts a gateway as trusted, we can derive a key from the AS PISKES key to authenticate the gateway too. Once the gateways can

[1] Future implementations will allow more than one interface so that the controller can act as a switch as well

communicate over an authenticated channel, they can perform a Diffie-Hellman (DH) key exchange to agree on a shared secret key to encrypt messages. The secure channel establishment process is the following (as shown in Fig. 5.2)[2]:

1. AS A retrieves the PISKES key $K_{A \to B}$ to authenticate itself from the local certificate server[3], and then,
2. it generates a random secret S_A and sends $DH_A = g^{S_A} \bmod p$ to B, along with a signature of DH_A using $K_{A \to B}$.
3. Upon reception of DH_A, B retrieves the PISKES key $K_{A \to B}$ to check the authenticity of the message, and then,
4. it generates a random secret S_B and sends $DH_B = g^{S_B} \bmod p$ to A along with a signature of DH_B, again using $K_{A \to B}$ to sign it.
5. Finally, both A and B compute the shared secret $S = (DH_B)^{S_A}\ mod\ p = g^{DH_B * DH_A}\ mod\ p = g^{DH_A * DH_B}\ mod\ p = (DH_A)^{S_B}\ mod\ p$

After the key exchange, each subsequent message between the gateways will be authenticated, with MAC($K_{A \to B}$, pld), and encrypted, with ENC$_S$(pld). The cryptographic functions used to authenticate and to encrypt a message are CMAC and AES-CBC with PKCS7 [Kal98] padding, respectively.

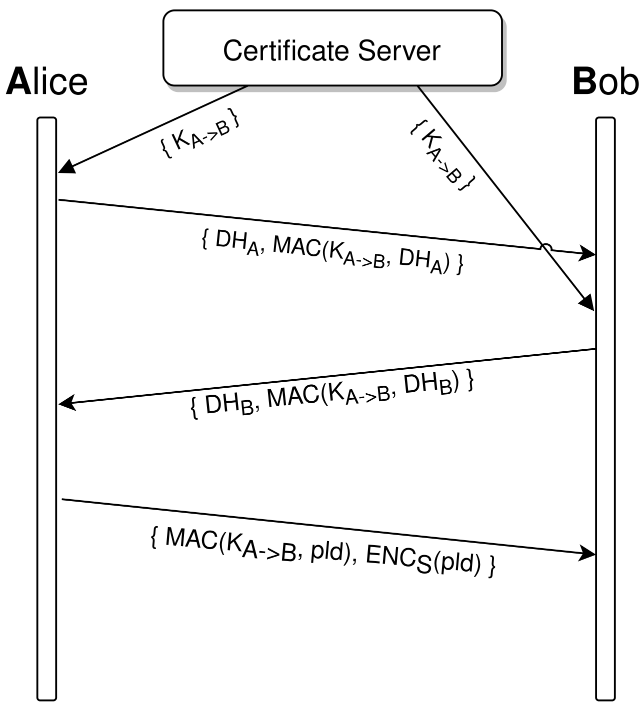


Figure 5.2: Secure channel establishment

[2] The generator g and the modulo p used for the DH key exchange are part of the gateways' configuration, hence previously agreed

[3] Note that the arrow only means that the key derivation has started from $K_{A \to *}$

5.3 Hidden Paths

As discussed in Section 3.3.1, *hidden paths* help to achieve availability guarantees by providing network-level access control. Fig. 5.3 depicts a simple SCION ISD, and we can use it to explain how hidden paths work. With some creativity, ASes A(lice) and B(ob) want to establish a hidden path, and E(ve) wants to DoS the communication between the two. The arrows in Fig. 5.3 represent the parent/child relationship between ASes.

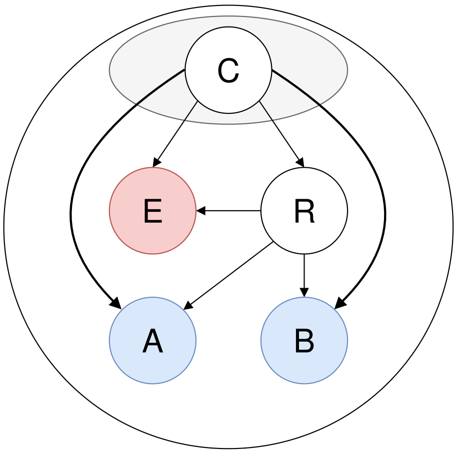


Figure 5.3: A simple ISD with four ASes, where C is a core AS

5.3.1 Designate hidden path

In the beginning, no paths are available, and the core of the ISD, composed of AS C, bootstraps the beaconing process for exploration. Following the arrows, the PCBs are the following: (C $\rightarrow$ E), (C $\rightarrow$ R), (C $\rightarrow$ A), (C $\rightarrow$ B), (C $\rightarrow$ R $\rightarrow$ E), (C $\rightarrow$ R $\rightarrow$ A), (C $\rightarrow$ R $\rightarrow$ B). Each received PCB, depending on the up-segment policy, is stored locally as an up-segment (in reverse order as received). At this point, no public down-segment is available, and no AS is reachable. ASes A and B can now use (A $\leftrightarrow$ R $\leftrightarrow$ B) as a hidden path by privately sharing the up-segments (A $\rightarrow$ R $\rightarrow$ C) and (B $\rightarrow$ R $\rightarrow$ C). The publicly registered down-segments will then only be: (C $\rightarrow$ E), (C $\rightarrow$ R), (C $\rightarrow$ A), (C $\rightarrow$ B). This way, both ASes A and B are still reachable by anybody in the ISD and from other ISDs. However, an adversary will not be able to access the hidden path shared by A and B.

5.3.2 Keeping the path secret

Although there is already a specification for hidden paths in SCION, the infrastructure was not updated to the most recent version, and it cannot be used. Nevertheless, we can still build a simple, in terms of integration with the rest of the system, hidden paths service that provides the same guarantees of the canonical one. We start by applying a down-segment policy to stop the path servers of A and B from registering down-segments including AS R. The following policy would be sufficient: `Filter: {AsBlackList: ["R"]}`.

5.3.3 Share segments

At this point, we need to share the up-segments to reach a rendezvous server, which in the example is represented by AS R. We start by retrieving the up path from the local path server. SCION offers a service called *scion daemon* that is similar to an Internet Domain Name System (DNS) resolver: given a destination IA, which is a combination of the ISD and AS numbers, it returns a set of paths to reach it. We could then use it to obtain the path to reach R; however, the information returned by the scion daemon API is not rich enough for our needs. We then instead directly query the database of the local path server to obtain a raw up-segment that leads to R. Once the two gateways gather a compatible up-segment, they need to share it with the other gateway. However, since these segments are the ultimate capability token that allows an AS to send traffic over the hidden path, they must be shared in a secure way to preserve their confidentiality. Otherwise, if E(ve) happens to be on-path, they could read the up-segments and later combine them to DoS the connection – which is in the attacker's capability as stated in Section 3.1.1.

Two channels can offer a secure exchange of such segments:

1. an out-of-band channel, e.g., by physically installing the paths on the gateways – which could be inconvenient if the gateways happen to be geographically far, or

2. an authenticated and confidential channel.

Fortunately, we can use the secure channel that we built in Section 5.2 to exchange the segments.

5.3.4 Build the hidden path

Upon reception of the segment, each AS then stitches it with one of its path segments using the `Combine(...)` method from the *combinator* package of the SCION library. The method takes as input a source s, a destination d, and a list of segments $\mathcal{S}$. The result is a list of all the paths (if any) that connect s to d. Paths are composed of hops contained in $\mathcal{S}$. Finally, A(lice) and B(ob) can communicate over the hidden path, and they have a cryptographic assurance that E(ve) cannot use that path. E(ve) could still attempt to DDoS the communication but they would be by the network infrastructure.

5.4 Fast Failover

Guaranteeing access means also taking into account unpredictable network failures, and putting in place back up connections is not enough. A network must be able to carry out a "fast failover", which consists of seamlessly shifting traffic from one connection to another. SCION multipathing (see Section 2.2.2) makes it easy to both establish back up connections and perform a fast failover among them. Again, we refer to Fig. 5.3 to make an example. ASes A and B connect with the path (A $\leftrightarrow$ R $\leftrightarrow$ B) and want to use a secondary path as a backup connection in case of a network failure: path (A $\leftrightarrow$ C $\leftrightarrow$ B) can serve this purpose. To avoid incurring extra latency to retrieve the backup path in case of failure, each gateway stores a pool of paths when instantiating a connection with the other gateway.

5.4.1 Failure Detection

When a link fails, the SCION infrastructure can inform an end-host about the failure by sending it a 'path revocation' using the Scion Control Message Protocol (SCMP), analogous to the Internet Control Message Protocol (ICMP). However, the current timeout, before sending a path revocation, is globally set to 3 s, which does not meet our requirements to perform a fast failover. We then implement failure detection system at the application level based on keepalive messages. Path failure detection can be tweaked in the `pathing` section of the configuration (listed in Fig. A.1) to account for various link qualities and preferences. The settings are the following:

- `keepalivetimeout`: the time delta from the last received keepalive message to declare a path inactive. This setting accounts for the *latency* of the connection,
- `keepaliveinterval`: the pace at which the gateway transmits keepalive messages. This setting can be used to account for *link quality*: if packets tend to be dropped from time to time on a link, but we still want to use it, lowering the `keepaliveinterval` can compensate this packet loss,
- `keepalivetimeoutinterval`: the pace at which the gateway checks for timeouts expiration. Its value should be `keepaliveinterval`/2, not to lose any keepalive message,
- `migrategracetimeout`: the first `keepalivetimeout` after a path migration.

Avoiding false positives

Failures might get detected at different times by gateways. With a very small keepalive timeout, the gateway that detects a failure first might consider a backup path as inactive, since it does not receive keepalive messages. In reality, the other gateway just discovered the loss later. The `migrategracetimeout` prevents the "faster" gateway from prematurely considering the new path as inactive.

We must note that a higher grace timeout avoids false positives, but on the other hand, it makes the system less responsive in case of multiple paths failures – which is, however, less likely than a false positive due to a small keepalive timeout. The goal of the grace timeout is to provide an administrator with the chance to find the sweet spot between the two.

Avoiding false negatives

Symmetrically to incurring in a false positive, the failure detection system could produce a false negative, considering a path still active while it is not. One of the two gateway might detect a link failure, migrate to another path, and restart sending keepalive messages before the timeout expires on the other gateway, which will still consider the path as being alive. To account for this issue, while maintaining the keepalive system simple (without adding, for example, an hinting system to inform the other gateway that a migration happened), we can perform a simple modification to the system: a gateway does not start sending keepalive messages right after a migration, but rather, it waits a `keepalivetimeout` period. Combined with an appropriate grace timeout, this modification makes it possible for the "slower" gateway to detect a failure despite a lack of time synchronization. Please note that this does not impact the data plane, which restarts sending packets right after a migration.

5.5 Legacy Compatibility: a SCION VPN

We can adapt the LaNeCa traffic relaying infrastructure to tunnel IP traffic, similar to what SCION-IP Gateway (SIG) does. The goal is not to build a replacement for the SIG, but rather, to experiment with SCION features not offered by the SIG, such as hidden paths and PISKES backed E2E encryption. We could have forked the SIG to include both these new SCION features and LaNeCa specific operations, but due to its complexity, it was easier to develop a new solution from scratch.

The `ip` section of the settings is straightforward:

1. `addr`: the IP address of the *tun* interface created by the gateway to listen for traffic, and
2. `subnets`: a list of the local subnets under the control of the gateway. When two gateways establish a connection, among the exchanged configurations, each gateway sends the other which subnets are under its control. Upon reception of remote subnets, each gateway creates a local route to intercept the traffic directed to them.

For instance, the gateway in AS A communicates the gateway in AS B that it controls the subnet `192.168.3.0/24`. The gateway in B then creates a rule like `192.168.3.0/24 dev tun0`, with `tun0` being the tun interface established by the gateway. This automatic traffic redirection could indeed introduce a security risk since the other gateway could hijack traffic. If this could still be a problem even though gateways are under the control of the same entity, a simple whitelisting of the allowed subnets could be added.

6 Evaluation

In this chapter, we evaluate the security and performance of LaNeCa and the deployability of SENECA. For the former, we analyze possible attacks on the smart factory use case from Section 4.4.1 and investigate the overhead of LaNeCa on two IoT boards. For the latter, we realize an IoT environment between Magdeburg (Germany) and Zurich (Switzerland) to assess the real-life challenges of a SENECA deployment.

6.1 LaNeCa Security

In this section, we analyze the most promising attacks that an adversary could perform to compromise the security of the smart factory LaNeCa deployment.

6.1.1 Compromising devices

An attacker can try to recover the entire root key set to be able to forge authorized packets. To recover the keys that compose the root key set, an adversary must compromise devices in multiple equivalence security classes. We can evaluate the likelihood of a successful attack if an adversary manages to compromise t security equivalence classes. In the use case of Section 4.4.1, we assign root keys to security equivalence classes based on the outcome of the sampling of a random variable $X \sim \text{Bern}(0.3)$. Each security equivalence class then receives a root key with probability $p = 0.3$ (and does not get it with probability $1 - p$). Compromising t security equivalence classes is like having t chances to sample 1 from X, which in other words means "not missing t chances to sample 1". Given that the sampling for every security equivalence class is independent, the probability of recovering a key by compromising t devices equals to

$$P_{\text{key}} = 1 - (1 - p)^t \tag{6.1}$$

Given also that the sampling for each root key is independent, the probability of compromising the entire root key set is

$$P_{\text{set}} = (1 - (1 - p)^t)^n \tag{6.2}$$

If the adversary manages to compromise even $t = 3$ security equivalence classes, which could mean compromising hundreds of devices out of ∼500 total devices, the attack would still have a success rate of less than 1.5%.

6.1.2 Brute-Forcing the First Hop

An adversary might still manage to slow down the network infrastructure by overcoming the first hop. In this case, the content of the packet is not relevant since the only goal is to flood the network with traffic.

In expectation, 3 out of 10 MACs are verified at the first hop, like in every other one. An adversary can then proceed in two ways: (i) assuming it manages to obtain a valid packet, it could attempt to break 3 AES ciphers, which is still be unfeasible [bre20], or (ii) they can brute-force the 3 16 bits MACs verified by the first hop. The last attack looks promising to overcome the first hop. We must note, however, that the attacker cannot test the 3 MACs separately for correctness, since the packet is either dropped or not. Moreover, they must also place all the MACs in the right order, for the same reason as above. The number of attempts needed to brute-force the part of the tag checked by the first hop is then not only $2^{16} \times 3$, but rather, $2^{48} \times \binom{10}{3} 3! = 2^{48} \times 720$, where 48 is the number of bits that need to be brute-forced, and 720 are the different ways of placing the 3 MACs. Moreover, to perform an attempt, the attacker must send a packet over the network. Assuming the smallest LaNeCa packet, hence 38 Bytes, and a whopping 10 Gbps connection between the attacker and the first hop, which we approximate for excess with 2^{31} Byte/s, the attack would take more than 113 years:

$$\frac{2^{48} \times 720 \times 38}{2^{31}} \mathrm{s} = 3586129920 \text{ s} \simeq 113 \text{ years} \tag{6.3}$$

6.2 LaNeCa Performance

In this section, we evaluate the overhead of a LaNeCa communication compared to that of pure Ethernet.

6.2.1 Benchmarking Setup

For the sake of experiments reproducibility, we present the benchmarking setup used to evaluate LaNeCa.

Microcontrollers boards Our implementation targets two ST Nucleo-F439ZI development boards, equipped with an STM32F439ZI MCU that runs at 168MHz, representing mid-range IoT devices. The boards also carry hardware accelerated crypto module - a pretty default feature on IoT MCUs, used to support protocols that require link-layer encryption (e.g., Wi-Fi Protected Access (WPA)). The connection is offered via an onboard 10/100 Mbps Ethernet interface.

Experiments Methodology The boards were directly connected via a 0.5 meter Ethernet cable CAT5e, and all the tests lasted 5 s. A note about variance in the tests: the results of the tests, across various repetitions, are the same up to packet precision.

Although this might look like an error, the cause is little entropy in each experiment. Given the precision of our measurements, there is almost no unpredictable external factor that can vary a test, which leads to extremely reproducible results with a minuscule (if not nil) variance.

Measuring Time Like in most benchmarks, we need an accurate measurement of time. However, on embedded systems, this easier said than done. Out of the box, the ST library provides a timer with a 1 ms resolution. While this precision might be enough to measure the duration of a test (1 ms out of 5 s is less than 0.1% of the total duration), this is not true for measuring code execution. We then use an external counter that we clock at 1 MHz (1×10^6 Hz) to have a resolution of 1 µs (1×10^{-6} s).

Benchmarking Framework From the very beginning of the development, we performed benchmarks to steer our future design choices to optimize the system. To make benchmarking more scalable and profitable, we developed a framework to run tests on the boards and audit the performances and correctness of the system. The framework orchestrates the boards via their serial interface. Also, to avoid interfering with the tests, all the measurements are collected locally and outputted at the end. The benchmarking framework takes care of the entire preparation, execution, and analysis pipeline.

Code Preparation and Deploy For performance and space reasons, LaNeCa's configuration gets defined at compile time. Testing a mode of operation then requires changing the source code, which is time-consuming and error-prone if performed manually. The framework, thus takes care of modifying the source code, compiling a binary, and flashing it on the boards.

Environment Setup Before starting a new benchmark, the framework restarts the boards, sets them in benchmarking mode, and wait for them to be ready to execute a test.

Test Start Synchronization Since the tests are time-based, synchronization in launching them is paramount. For this reason, the framework does not directly start a test. Instead, it is the sender that with the first transmitted packet signals the receiver the beginning of a test.

Data Extraction and Visualization At the end of every run (e.g., every new payload size), each board prints on the serial line the results of the benchmark. Results are then parsed and aggregated in a benchmark dictionary that is processed and visualized.

6.2.2 Computing MACs Efficiently

Performance Predictions

To further confirm the results of the benchmarks that we perform in the next sections, we can first carry out an "a priori" prediction of how the Plain and Pre-Hash mode will perform. We do this by looking at the crypto primitives involved in the computation of each of m MACs over a payload of size n. $F[l]$ indicates that we invoke function F with an input of l bytes. The cost of the Plain mode computation is:

$$C_{\text{Plain}}(n) = \text{AES}[max(0, n - \text{AES}_{\text{BLOCK}})] + \text{AES}[\text{AES}_{\text{BLOCK}}] \tag{6.4}$$

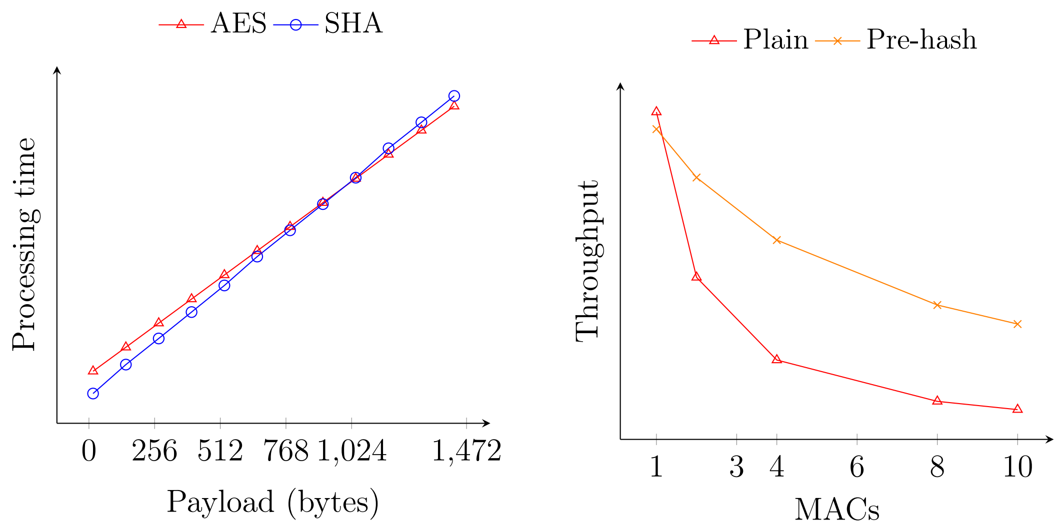
The cost of the Pre-hashing mode computation is:

$$C_{\text{Pre-hash}}(n) = \frac{\text{SHA}[n]}{m} + \text{AES}(2 \times \text{AES}_{\text{BLOCK}}) \tag{6.5}$$

To predict the performance of each mode we define a throughput trend function:

$$T_{\text{mode}}(n, m) = \frac{1}{m \times C_{\text{mode}}(n)} \tag{6.6}$$

Figure 6.1a [1]shows the empirically gathered processing time of AES-128 and SHA-256. Plugging in this results in T_{Plain} and $T_{\text{Pre-hash}}$, we obtain a prediction of the throughput trend. Figure 6.1b [1] shows the results of $T_{\text{Plain}}(1476, 10)$ and $T_{\text{Pre-hash}}(1476, 10)$. Based on the results, we expect the Pre-hash mode to scale better with a growing number of MACs.



(a) Processing time trends of AES and SHA with growing payload

(b) Expected throughput trend of Plain and Pre-hash modes

[1]There is no scale in the y-axis since the graph only shows the trend of the function

Throughput Against Number of MACs

The predictions of the throughput trends turn out to be accurate, as shown in Fig. 6.2, further suggesting that the tests we correctly performed. The Pre-hash mode

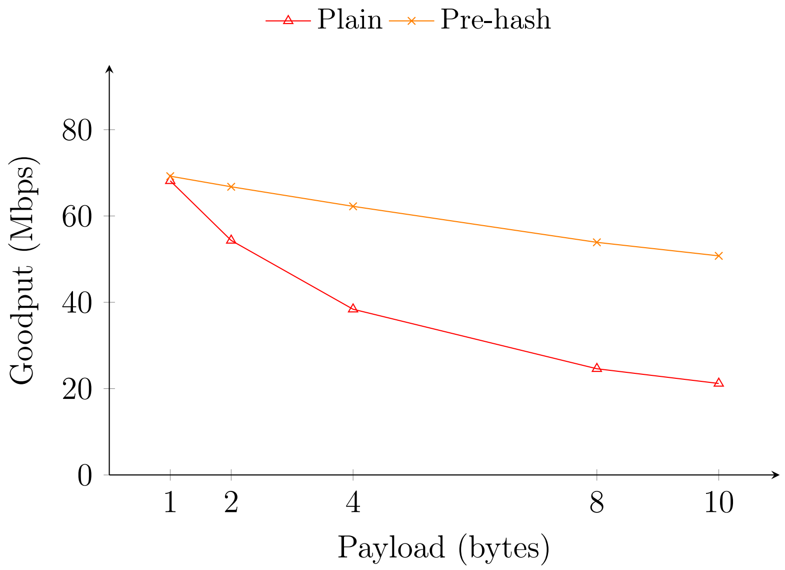


Figure 6.2: Plain vs Pre-hash goodput with increasing number of MACs

performs better than the Plain mode and also scales better to a growing number of tags. Moreover, unlike the prediction, the Pre-hash mode outperforms the Plain mode in every situation. The reason being that the prediction model C_{Plain} ignores some constants. Looking at load profiles in Figs. 6.3 and 6.4, we can further inspect the overhead of MACs computation.

As expected, the network share of time increases with bigger payloads. The reason for this is that the load profile, over a certain period, is a representation of the amount of time spent for each component involved in sending (and receiving) a message, since it is the only action performed by the system. When dealing with smaller packets, hence smaller payloads, the time the NIC needs to move data on the wire is modest, while the per-packet overhead, represented by all the other functions, is significant. On the other hand, with bigger packets, hence bigger payloads, the NIC will take more time to send them over the wire, and the per-packet overhead is smaller.

When using 1 MAC, the overhead is comparable for the two modes, with the functions that process the entire payload, SHA in one case, and AES in the other, covering most of the computational time. However, with 10 MACs, the situation changes: Pre-hashing the payload now covers less than the time spent for the computation of the MACs, and the difference is now significant. Also, the load profiles confirm how the hashing computation of the Pre-hashing mode gets amortized with an increasing number of MACs.

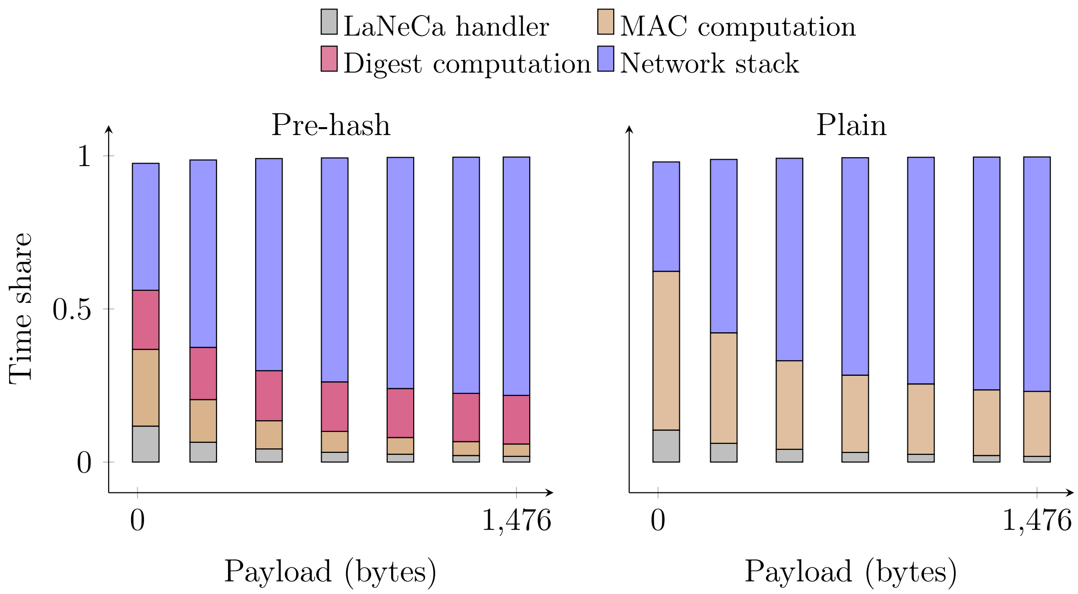


Figure 6.3: Simplified load profile (1 MAC)

6.2.3 Ethernet vs LaNeCa

In this subsection, we finally compare LaNeCa's overhead to plane Ethernet. Fig. 6.5 shows a comparison between Ethernet, LaNeCa with 1 MAC, and LaNeCa with 10 MACs. As the graph shows, LaNeCa is lacking the last data point corresponding to 1500 bytes – Ethernet maximum transmission unit (MTU), because of the additional information in the header of the packet. First, we notice that the Ethernet baseline manages to saturate the NIC, peaking at ≈90 Mbps. Then, LaNeCa with a single MAC reaches ≈68 Mbps, and LaNeCa with 10 MAC gets as high as ≈51 Mbps. The SW only implementation, which is not comparable to the HW accelerated counterpart, still reaches an acceptable ≈7.5 Mbps and ≈11 Mbps with 10 MACs and 1 MAC, respectively. Finally, looking again at the load profiles of the Pre-hash mode in Figs. 6.3 and 6.4, we can see that the time spent in the LaNeCa handler, which is comparable both for input and output, is minimal compared to the time spent by the cryptographic and the network operations.

6.3 SENECA

In this section, we present the hardware and software used to build a SENECA gateway, and the geographical positioning and network topology used in the real-life experiment to evaluate it.

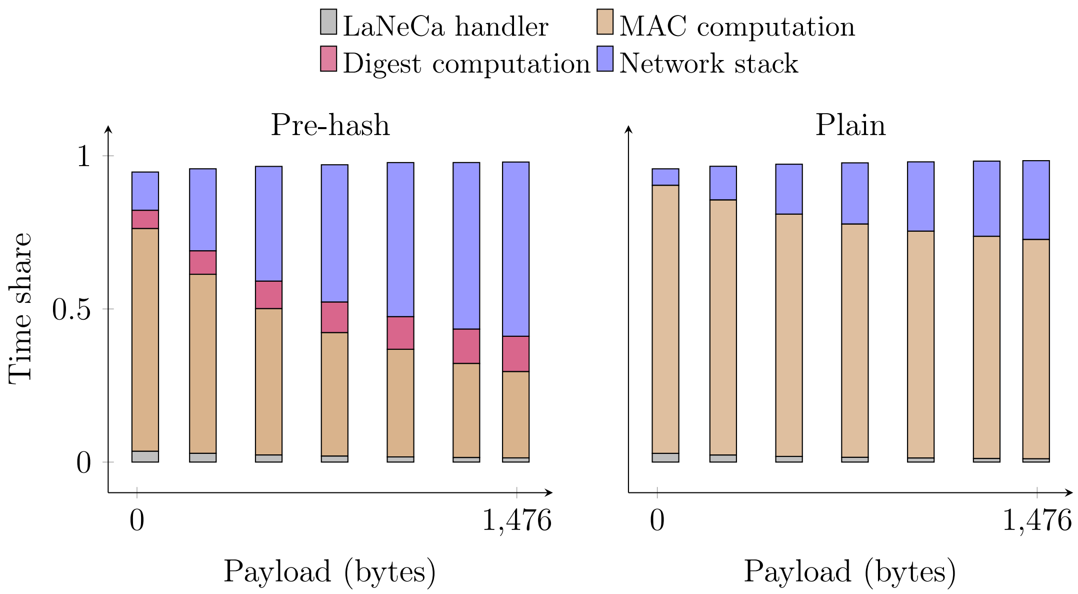


Figure 6.4: Simplified load profile mode (10 MACs)

6.3.1 Gateway Boxes

In line with using only off-the-shelf components, the network platform used for the gateway is the small PCEngine APU2 networking board[2]. The platform has 3 Ethernet NICs so we can easily experiment with back up physical links. The OS flashed on the boxes is OpenWRT[3], a Linux operating system targeting lightweight and embedded networking devices.

6.3.2 Geographical Displacement

We evaluate LaNeCa in a real-life scenario by putting in place an IoT station controlled from a remote C&C. We situate the C&C center in Magdeburg (Germany), and the IoT station in Zurich (Switzerland). Fig. 6.6 shows the locations of the two environments.

6.3.3 SCION Topology

Each domain, namely the C&C and the IoT station, has a box at its edge. Furthermore, a third box, placed in Zurich, acts as a hidden path rendezvous for the other two. Each one runs its own AS, namely 17-ffaa:1:daf, for the C&C in Magdeburg, 17-ffaa:1:c5f, for the IoT station in Zurich, and 17-ffaa:1:c3c, for the hidden path rendezvous, still in Zurich. To bootstrap the topology in Fig. 6.7, we used SCIONLab[4],

[2]More information the PCEngine APU2 is available at pcengines.ch
[3]More information about OpenWRT is available at openwrt.org
[4]More information about SCIONLab is available at scionlab.org

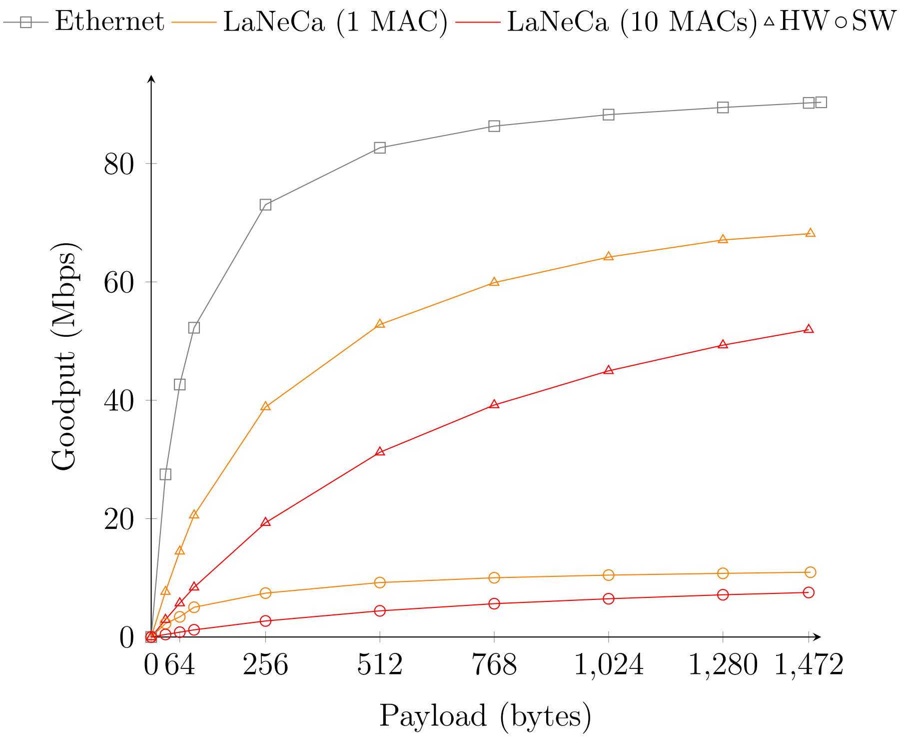


Figure 6.5: Ethernet vs. LaNeCa goodput performance.

the SCION testbed. Every box, then, is also connected to it. To keep the path with c3c hidden, daf and c5f use a down segment policy like the one in Section 5.3.2, swapping R with c3c.

6.3.4 An IoT Testbed

The IoT station, located in Zurich, is provided with off-the-shelf sensors and actuators, as shown in Fig. 6.8. The master controller connects to a Raspberry Pi, which in turn connected to the network box via Ethernet. There are two processes: a light process, with an LED matrix actuator and a light sensor, and a temperature process, with a Peltier device [5] actuator and a temperature sensor. A dual relay, which interfaces with a DC generator, controls the actuators. The sensors and the actuators are, respectively, queried and operated remotely from the C&C center.

6.3.5 Testing Failover

To analyze the efficiency of the failover, we inspect the impact of a link failure on communication between gateways. The keepalive configuration of the two gateways is the one shown in Fig. A.1. First, we evaluate the raw failover performances by

[5] More information on Peltier devices is available at en.wikipedia.org/wiki/Thermoelectric_cooling

Figure 6.6: The locations of the C&C () and the IoT station ()

performing a UDP test. Then, we inspect the effects of failover on a TCP-based application.

Iperf Results

We use the gateways to relay legacy IP traffic so we can use the Iperf3[6] tool. The test consists of transmitting UDP packets at 10 MBps. We start the Iperf3 server with a 1 ms sensing interval in the c5f AS with:

```
[17-ffaa:1:c5f]$ iperf3 -s -i 0.1
```

and the client in daf with:

```
[17-ffaa:1:daf]$ iperf3 -c 192.168.2.100 -u -b 10Mbps -t 3
```

While the test is running, we bring down the interface which connects the gateways with:

```
[17-ffaa:1:c5f]$ ip link set eth0 down
```

The result, Fig. 6.9, shows that the time windows affected are [1.30-1.40], [1.40-1.50], and [1.50-1.60] and that a total of 196 packets where lost. The sender transferred 3266 packets, one every $\approx$900 µs. Assuming a constant transfer rate, the total downtime experienced by the end application is 900×196 µs $\simeq 176$ ms.

Application Impact

We now inspect the impact of a link failure from an application perspective. Although the C&C center and the station are both in Zurich, we tunnel all the traffic via Magdeburg.

[6]More information on the Iperf3 tool is available at iperf.fr

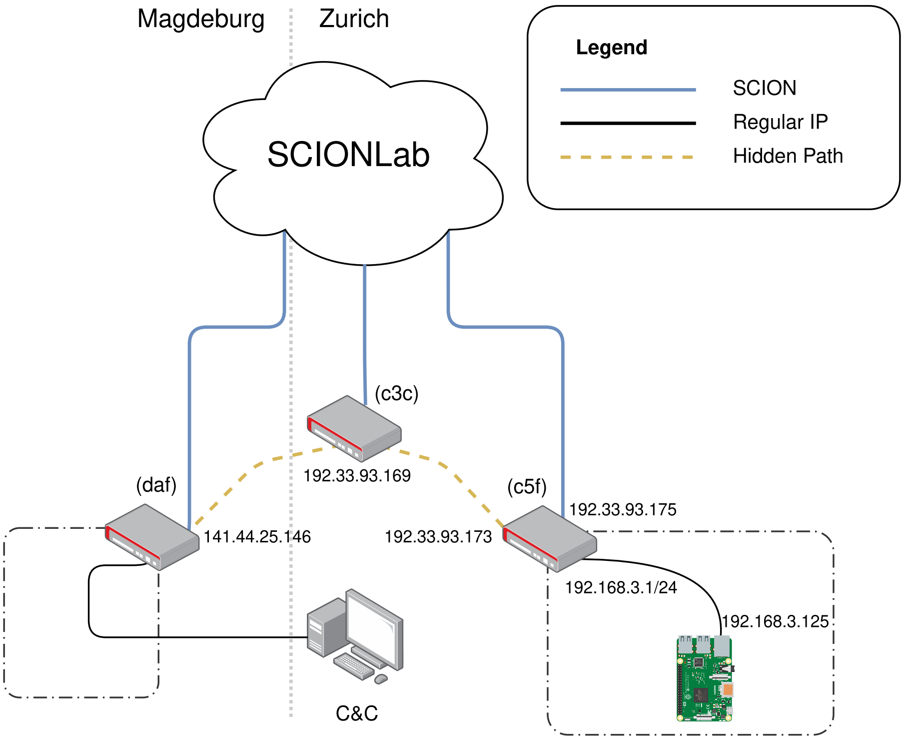


Figure 6.7: The network topology used between Magdeburg and Zurich

IP Setup We start with the IP setup, and we set up a tunnel with:

```
[C&C]$ ssh scion@141.44.25.146 -L42230:192.33.93.173:4223 -N
```

The command creates a simple Secure Shell (SSH) tunnel which relays the traffic directed to localhost:42230 to the box in Magdeburg, which in turn, sends it to 192.33.93.173:4223 – a port forwarding in the NAT, from 192.33.93.173:4223 to 192.168.3.125:4223, makes the Raspberry Pi available from the WAN. We connect to the Raspberry Pi, via the established tunnel, using a regular TCP connection, and plot sensors readings and periodically operate the actuators:

- the LED matrix is turned on for 2 s every 5 s, producing a sharp square wave Fig. 6.10a, and
- the Peltier device (thermoelectric cooler) activates for 1 s every 20 s, producing a smooth sinusoidal like shape Fig. 6.10b. This behavior is due to the thermal inertia of a heat sink attached to the Peltier device.

In between the third triggering of the light (not counting the first half-wave), we simulate a link failure/DDoS attack by physically bringing down the link which connects the two boxes. The connection is suddenly lost, and neither the readings nor the actuators are under control anymore – when the connection is lost the graphs keep plotting the latest reading.

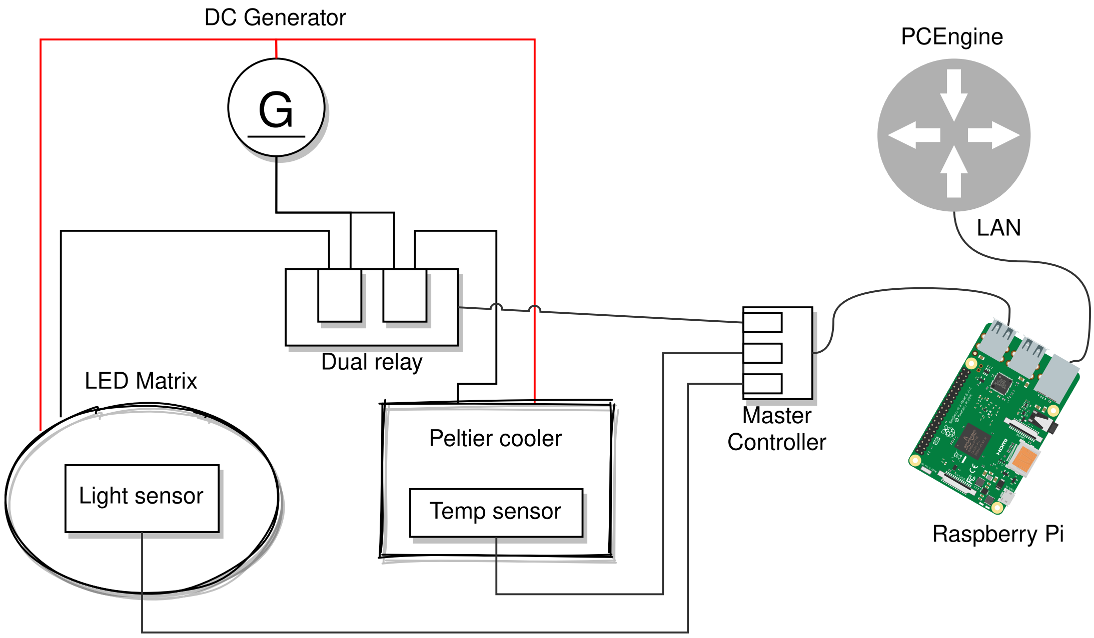


Figure 6.8: The IoT testbed

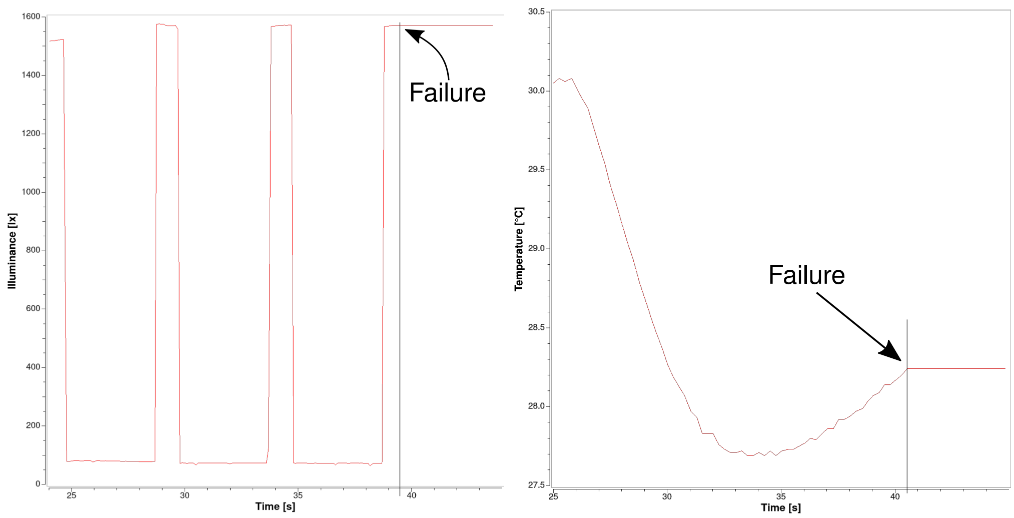
```
Interval         Transfer     Bandwidth         Jitter     Lost/Total
...
0.70-0.80 sec     122 KBytes  10.0 Mbits/sec   0.221 ms   0/109 (0%)
0.80-0.90 sec    56.1 KBytes  4.59 Mbits/sec   0.357 ms   31/81 (38%)
0.90-1.00 sec    0.00 Bytes   0.00 bits/sec    0.357 ms   0/0 (-nan%)
1.00-1.10 sec    90.8 KBytes  7.44 Mbits/sec   0.225 ms   165/246 (67%)
1.10-1.20 sec     122 KBytes  10.0 Mbits/sec   0.240 ms   0/109 (0%)
...
0.00-3.07 sec    3.58 MBytes  9.76 Mbits/sec   0.154 ms   196/3266 (6%)
```

Figure 6.9: Output of Iperf3 during a simulated link failure

SCION Setup We then perform the same test with the SCION setup. First, we start the gateways on boxes 17-ffaa:1:daf and 17-ffaa:1:c5f with the configurations in Figs. A.2a and A.2b. Then, we start a proxy to relay our traffic to Magdeburg:

```
[C&C]$ ssh scion@141.44.25.146 -L42230:192.168.3.125:4223 -N
```

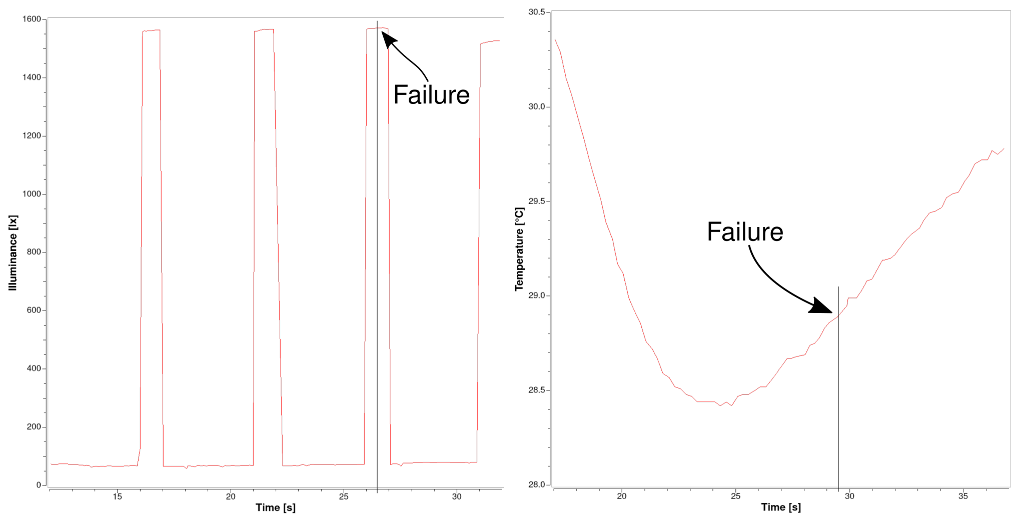
The proxy command is very similar to the one issued for the IP setup. However, unlike the previous one, the traffic is directed, from the box in Magdeburg, directly to the Raspberry Pi address, 192.168.3.125. The configuration in Fig. A.2b instructs the gateway in 17-ffaa:1:daf to intercept the traffic directed to 192.168.3.0/24. Packets targeted at 192.168.3.125 are intercepted relayed via SCION to 17-ffaa:1:c5f, which dispatches it locally to the Raspberry Pi. More precisely, in the beginning, the traffic follows the path 17-ffaa:1:daf ↔ 17-ffaa:0:1107 ↔ 17-ffaa:1:c5f – AS 1107 is



(a) Illuminance (b) Temperature

Figure 6.10: IP failure seen from sensors readings

an attachment point of SCIONLab. The interaction with the station is the same as in the IP setup, and like before, in between the third triggering of the light, we bring down the link which connects the two boxes. After 100 ms, each gateway discovers the failure and seamlessly shift the traffic onto another path. Since hidden pathing is enabled, the alternative route chosen by the gateways is 17-ffaa:1:daf ↔ 17-ffaa:1:c3c ↔ 17-ffaa:1:c5f. The efficiency of the fast failover is proven by Figs. 6.11a and 6.11b. Moreover, an attacker would not be able to attack the back-up hidden path.



(a) Illuminance (b) Temperature

Figure 6.11: SCION failover seen from sensors readings

7 Conclusion

7.1 Future research

We laid the foundations of LaNeCa, making its implementation highly adaptable. A first step to extend LaNeCa would be implementing it at L3, which would be a great addition to its overall adaptability. Moreover, encryption-by-default could be explored using authenticated encryption mechanisms, which would add confidentiality to the existing LaNeCa authentication scheme.

Another interesting direction would be extending the LaNeCa controller. For example, one could make it compliant with the Manufacturer Usage Description (MUD) standard.

Lastly, SENECA can be improved too, and the first step could be adding SIBRA to achieve even different availability guarantees.

7.2 Summary

In Chapter 1, we highlighted the urgency of securing the Internet of Things (IoT). We showed its ever-growing adoption and its current state of (in)security. In Chapter 3, we reviewed possible technologies to realize a network architecture to secure IoT environments. In Chapter 4, we investigated the realization of L2.5 LaNeCa, and in Chapter 5, we augmented it with SCION (SENECA) to handle across-domain traffic too. Finally, in Chapter 6, we performed benchmarks and security analyses to evaluate the efficiency and resilience of LaNeCa, and we realized a proof-of-concept to assess the advantages and deployability of SENECA.

7.3 Contribution

The contribution of this work is twofold: first, it systematically defines the services that a network must provide to an IoT environment to secure it. Second, it proposes (and implements) an network system that offers said services and can be deployed in a wide range of IoT environments and products.

We hope that this work will directly, by deploying our solution, and indirectly, by laying the foundation for other works, help developers and manufacturers to bring back the S, which stands for Security, in IoT.

Bibliography

[AAB+17] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, August 2017. USENIX Association.

[Axe15] C Warren Axelrod. Enforcing security, safety and privacy for the internet of things. In *2015 Long Island Systems, Applications and Technology*, pages 1–6. IEEE, 2015.

[Bar19] Chris Baraniuk. Anaesthetic devices 'vulnerable to hackers'. `https://www.bbc.com/news/technology-48935111`, 2019.

[BBB+06] Elaine Barker, William Barker, William Burr, William Polk, Miles Smid, et al. *Recommendation for key management: Part 1: General.* National Institute of Standards and Technology, Technology Administration, 2006.

[Blo20] Bloomberg. Internet of things (iot) market worth $1319.08 billion, globally, by 2026 at 25.68 `https://www.bloomberg.com/press-releases/2020-07-14/internet-of-things-iot-market-worth-1319-08-billion-globally-by-2026-at-25-68-cagr-verified-market-research`, 2020.

[bre20] How long would it take to brute force an aes-128 key? `https://crypto.stackexchange.com/questions/48667/how-long-would-it-take-to-brute-force-an-aes-128-key`, 2020.

[BRS+15] Cristina Basescu, Raphael M Reischuk, Pawel Szalachowski, Adrian Perrig, Yao Zhang, Hsu-Chun Hsiao, Ayumu Kubota, and Jumpei Urakawa. Sibra: Scalable internet bandwidth reservation architecture. *arXiv preprint arXiv:1510.02696*, 2015.

[CCS15] I. Coonjah, P. C. Catherine, and K. M. S. Soyjaudah. Experimental performance comparison between tcp vs udp tunnel using openvpn. In *2015 International Conference on Computing, Communication and Security (ICCCS)*, pages 1–5, 2015.

[CFP+07] Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking control of the enterprise. *ACM SIGCOMM computer communication review*, 37(4):1–12, 2007.

[CGA+06] Martin Casado, Tal Garfinkel, Aditya Akella, Michael J Freedman, Dan Boneh, Nick McKeown, and Scott Shenker. Sane: A protection architecture for enterprise networks. In *USENIX Security Symposium*, volume 49, page 50, 2006.

[CHB+13] Danny Cooper, Ethan Heilman, Kyle Brogle, Leonid Reyzin, and Sharon Goldberg. On the risk of misbehaving rpki authorities. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, pages 1–7, 2013.

[Clo17] Cloudflare. Famous ddos attacks | the largest ddos attacks of all time. `https://www.cloudflare.com/learning/ddos/famous-ddos-attacks/`, 2017.

[Clo20a] Cloudflare. Google goes down after major bgp mishap routes traffic through china. `https://arstechnica.com/information-technology/2018/11/major-bgp-mishap-takes-down-google-as-traffic-improperly-travels-to-china`, 2020.

[Clo20b] Cloudflare. Http flood attack. `https://www.cloudflare.com/learning/ddos/http-flood-ddos-attack`, 2020.

[Clo20c] Cloudflare. What is bgp hijacking? `https://www.cloudflare.com/learning/security/glossary/bgp-hijacking/`, 2020.

[Dun01] Adam Dunkels. Design and implementation of the lwIP TCP/IP stack. *Swedish Institute of Computer Science*, 2(77), 2001.

[Gar20a] Gartner. Gartner says 5.8 billion enterprise and automotive iot endpoints will be in use in 2020. `https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-io`, 2020.

[Gar20b] Gartner. Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016. `https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016`, 2020.

[Gar20c] Gartner. Sd-wan is killing mpls, so prepare to replace it now. `https://www.gartner.com/en/documents/3890682/sd-wan-is-killing-mpls-so-prepare-to-replace-it-now`, 2020.

[Gre13] Matthew Green. Why i hate cbc-mac. `https://blog.cryptographyengineering.com/2013/02/15/why-i-hate-cbc-mac/`, 2013.

[Gá19] J. J. Gálvez. Hackers broadcast live stream of police camera at podemos leaders' home. https://english.elpais.com/elpais/2019/04/08/inenglish/1554706393_909358.html, 2019.

[HH18] Chung-Wen Hung and Wen-Ting Hsu. Power consumption and calculation requirement analysis of AES for WSN IoT. *Sensors*, 18(6):1675, may 2018.

[IEE20a] IEEE. 802.1q-2014 | bridges and bridged network. http://www.ieee802.org/1/pages/802.1Q-2014.html, 2020.

[IEE20b] IEEE. Ieee standard for an architectural framework for the internet of things (iot). *IEEE Std 2413-2019*, pages 1–269, 2020.

[Int17] InternetSociety. 14,000 incidents: A 2017 routing security year in review. https://www.internetsociety.org/blog/2018/01/14000-incidents-2017-routing-security-year-review/, 2017.

[Kal98] Burt Kaliski. Pkcs #7: Cryptographic message syntax version 1.5. RFC 2315, RFC Editor, March 1998. http://www.rfc-editor.org/rfc/rfc2315.txt.

[KKK07] Nate Kushman, Srikanth Kandula, and Dina Katabi. Can you hear me now?! it must be bgp. *SIGCOMM Comput. Commun. Rev.*, 37(2):75–84, March 2007.

[KM18] Ronny Ko and James Mickens. Deadbolt: Securing iot deployments. In *Proceedings of the Applied Networking Research Workshop*, pages 50–57, 2018.

[LABJ01] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed internet routing convergence. *IEEE/ACM Transactions on Networking*, 9(3):293–306, 2001.

[LHZ14] Qi Li, Yih-Chun Hu, and Xinwen Zhang. Even rockets cannot make pigs fly sustainably: Can bgp be secured with bgpsec. In *Workshop SENT'14, 23 February 2014, San Diego, USA, Copyright 2014 Internet Society: Proceedings*. Internet Society, 2014.

[MMH+17] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. Iot sentinel: Automated device-type identification for security enforcement in iot. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2177–2184. IEEE, 2017.

[New20] Lily Hay Newman. Hackers could use iot botnets to manipulate energy markets. https://www.wired.com/story/hackers-iot-botnets-manipulate-energy-markets, 2020.

[NIS05] NIST. *Recommendation for Block Cipher Modes of Operation: the CMAC Mode for Authentication*, May 2005.

[NISft] NIST. *Recommendations for IoT Device Manufacturers: Foundational Activities and Core Device Cybersecurity Capability Baseline*, 2020 (Second Draft).

[O'D19] Lindsey O'Donnell. Unpatched flaws in iot smart deadbolt open homes to danger. `https://threatpost.com/unpatched-flaws-in-iot-smart-deadbolt-open-homes-to-danger/146871/`, 2019.

[OH04] Hilarie Orman and Paul Hoffman. Determining strengths for public keys used for exchanging symmetric keys. Technical report, BCP 86, RFC 3766, April, 2004.

[OMM17] Yasuhisa Omura, Abhijit Mallik, and Naoto Matsuo. *MOS Devices for Low-voltage and Low-energy Applications*. John Wiley & Sons, 2017.

[OZPZ09] R. Oliveira, B. Zhang, D. Pei, and L. Zhang. Quantifying path exploration in the internet. *IEEE/ACM Transactions on Networking*, 17(2):445–458, 2009.

[QPP+17] D. Quarta, M. Pogliani, M. Polino, F. Maggi, A. M. Zanchettin, and S. Zanero. An experimental security analysis of an industrial robot controller. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 268–286, 2017.

[RRLP20] Benjamin Rothenberger, Dominik Roos, Markus Legner, and Adrian Perrig. Piskes: Pragmatic internet-scale key-establishment system. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (ASIA CCS'20)*, 2020.

[SHNN12] Matti Siekkinen, Markus Hiienkari, Jukka K. Nurminen, and Johanna Nieminen. How low energy is bluetooth low energy? comparative measurements with ZigBee/802.15.4. In *2012 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. IEEE, apr 2012.

[SOM14] F. Shrouf, J. Ordieres, and G. Miragliotta. Smart factories in industry 4.0: A review of the concept and of energy management approached in production based on the internet of things paradigm. In *2014 IEEE International Conference on Industrial Engineering and Engineering Management*, pages 697–701, 2014.

[SPLI06] JH. Song, R. Poovendran, J. Lee, and T. Iwata. The aes-cmac algorithm. RFC 4493, RFC Editor, June 2006.

[SVG13] Y. Song, A. Venkataramani, and L. Gao. Identifying and addressing protocol manipulation attacks in "secure" bgp. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 550–559, 2013.

[Sym20] Symantec. An internet of things reference architecture. https://docs.broadcom.com/docs/iot-security-reference-architecture-en, 2020.

[TC20] Charles Towers-Clark. Iot security must evolve to survive. https://www.forbes.com/sites/charlestowersclark/2019/06/14/iot-security-must-evolve-to-survive, 2020.

[Tim20] New York Times. In the rush to join the smart home crowd, buyers should beware. https://www.nytimes.com/2019/01/22/business/smart-home-buyers-security-risks.html, 2020.

[Tor19] Georgina Torbet. Insulin pumps recalled for vulnerability; concerns raised over medical iot hacks. https://www.digitaltrends.com/computing/insulin-pump-hack-medical-iot/, 2019.

[Tre19] Joseph Trevithick. The u.s. army's new up-gunned stryker armored vehicles have been hacked. https://www.thedrive.com/the-war-zone/26458/the-u-s-armys-new-up-gunned-stryker-armored-vehicles-have-been-hacked, 2019.

[Wik19] Wikipedia contributors. Cbc-mac — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=CBC-MAC&oldid=926738709, 2019. [Online; accessed 10-August-2020].

[Wik20a] Wikipedia contributors. Anycast — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Anycast&oldid=970795468, 2020. [Online; accessed 22-August-2020].

[Wik20b] Wikipedia contributors. Bgp hijacking — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=BGP_hijacking&oldid=964360841, 2020. [Online; accessed 22-August-2020].

[Wik20c] Wikipedia contributors. Double-ended queue — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Double-ended_queue&oldid=962716369, 2020. [Online; accessed 10-August-2020].

[Wik20d] Wikipedia contributors. Edge computing — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Edge_computing&oldid=971671253, 2020. [Online; accessed 22-August-2020].

[Wik20e] Wikipedia contributors. Syn flood — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=SYN_flood&oldid=971688103`, 2020. [Online; accessed 22-August-2020].

A Figures

```
local: {address: A,127.0.0.1, ctrlport: 23000, dataport: 23001}
remotes:
  - {address: B,127.0.0.1, ctrlport: 23000, dataport: 23001,
   ↪  rendezvous: R}
laneca:
  interface: br-lan
  macs: ['00:00:00:00:00:0A']
  policies: ...
ip: {addr: 192.168.3.100, subnets: [192.168.3.0/24]}
pathing: {keepalivetimeout: 100ms, keepaliveinterval: 30ms,
 ↪  keepalivetimeoutinterval: 10ms, migrategracetimeout: 300ms}
```

Figure A.1: A gateway configuration file example

```
local:
  address:
   ↪  17-ffaa:1:daf,127.0.0.1,
  ctrlport: 23000,
  dataport: 23001
remotes:
  - address:
   ↪  17-ffaa:1:c5f,127.0.0.1,
    ctrlport: 23000,
    dataport: 23001,
    rendezvous: 17-ffaa:1:c3c
ip:
  addr: 192.168.2.100,
  subnets: [192.168.2.100/32]
```

(a) Configuration for 17-ffaa:1:daf

```
local:
  address:
   ↪  17-ffaa:1:c5f,127.0.0.1,
  ctrlport: 23000,
  dataport: 23001
remotes:
  - address:
   ↪  17-ffaa:1:daf,127.0.0.1,
    ctrlport: 23000,
    dataport: 23001,
    rendezvous: 17-ffaa:1:c3c
ip:
  addr: 192.168.3.100,
  subnets: [192.168.3.0/24]
```

(b) Configuration for 17-ffaa:1:c5f

Figure A.2: Configuration for the IoT Testbed

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

AN ALL-ROUND SECURE IOT NETWORK ARCHITECTURE

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
TULIMIERO	ANDREA

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zurich, 11/09/2020	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.