Automating VPN Configuration in Computer Networks

Automating VPN configuration in computer networks

Daniele Bringhenti, Riccardo Sisto, Fulvio Valenza

***Abstract*—The configuration of security systems for communication protection, such as VPNs, is traditionally performed manually by human beings. However, because the complexity of this task becomes soon difficult to manage when its size increases, critical errors that may open the door to cyberattacks may be introduced. Moreover, even when a solution is computed correctly, sub-optimizations that may afflict the performance of the configured VPNs may be introduced. Unfortunately, the possible solution that consists in automating the definition of VPN configurations has been scarcely studied in literature so far. Therefore, this paper proposes an automatic approach to compute the configuration of VPN systems. Both the allocation scheme of VPN systems in the network and their protection rules are computed automatically. This result is achieved through the formulation of a Maximum Satisfiability Modulo Theories problem, which provides both formal correctness-by-construction and optimization of the result. A framework implementing this approach has been developed, and its experimental validation showed that it is a valid alternative for replacing time-consuming and error-prone human operations for significant problem sizes.**

***Index Terms*—VPN, network security, policy-based management**

I. Introduction

Recently, preventing network traffic from undesired inspections and alterations has become a crucial security requirement, due to higher privacy needs and the incessant growth of hijacking attacks. The main mechanism for performing this task is the creation of *Virtual Private Networks* (VPNs). The traffic crossing VPNs is protected by systems located at the VPN's border, also called *Communication Protection Systems* (CPSs), which may be VPN gateways or the end points of the communication. Of course, a correct configuration of CPSs is required to guarantee an effective protection from cyber attacks, e.g., to avoid that a certain traffic can cross an untrustworthy part of the network without encryption. At the same time, there may be requirements to have a certain traffic unencrypted in some nodes of the network, e.g., for monitoring purposes.

Unfortunately, critical errors and sub-optimizations often afflict the configuration of these systems [1]. These issues are mainly originated by the traditional practice of configuring CPSs manually. The complexity and variety of communication protection solutions make this task error prone even with relatively small sizes of the problem. As evidence, an empirical assessment of the problem [2] showed that more than 90% of the security administrators participating in the study introduced at least one anomaly in the communication protection configuration of a relatively small network. Additionally, the most worrisome concern deriving from this study is that even people with a high expertise in the security field were involved and still introduced anomalies.

The anomalies deriving from a manual VPN configuration may afflict two different aspects: the allocation of CPSs in the network topology, and the definition of their protection rules.

About allocation, deciding the correct and best positions in the network topology where CPSs should be allocated is quite challenging for humans, because of the large variety of usage modes for VPNs (e.g., securing a traffic with an end-to-end VPN, creating a tunnel with a site-to-site VPN, or allowing secure accesses with a remote-access VPN). Therefore, a manually defined allocation scheme may result incorrect or sub-optimized. For instance, if a CPS is allocated too far from the source of a communication that must be protected, it is possible that in the path from that source to the CPS the traffic can be inspected or modified. Sub-optimizations, instead, may occur when a redundant number of CPSs is allocated in the topology, which may lead to a larger amount of traffic subject to encryption or hashing algorithms, and, consequently, a decreased global efficiency of the network forwarding operations.

About protection rule definition, the large variety of available cipher suites for protecting communications is a reason why anomalies are often introduced when defining such rules. These anomalies can be classified as errors, sub-optimizations and conflicts, and they can arise among the rules of a single CPS (intra-function anomaly) or among rules of different CPSs (inter-function anomaly) [2]. All such anomalies can be exploited by attackers to undermine the communication protection in the network or decrease network efficiency. For example, an incorrect rule definition may lead to problems such as the skewed channel, where traffic that is expected to be encrypted is actually plain, or the creation of site-to-site VPNs whose traffic crosses untrusted network areas.

Guaranteeing that the results of both operations are error-free and optimized would be a benefit for a large variety of different scenarios. For example, service providers would like to design their own physical networks, deciding where CPSs should be positioned and how they should be configured to satisfy their communication protection requirements. A similar problem may occur with virtualized networks [3] [4], where network functions can be deployed dynamically on general-purpose servers. In this case, service providers may want to dynamically define the position and configuration of CPSs in

D. Bringhenti, R. Sisto, and F. Valenza are with the Politecnico di Torino, Dip. Automatica e Informatica; e-mail: {first.last}@polito.it.

their virtual network so as to satisfy security requirements requested by their customers.

In view of all these considerations, this paper focuses on the problem of automating VPN configuration. In order to address all the main issues illustrated above, a security automation approach should be able to compute both the allocation scheme and the rule sets of the necessary CPSs, through an automatic policy refinement process, in which security administrators just have to provide information about the network itself (without CPSs) and the security policies to be enforced. Other two desirable requirements are the achievement of a formal correctness assurance about the refinement result and the selection of the optimal (e.g., the most efficient) solution among all the correct ones. Our proposal aims to fill this gap focusing on VPNs based on the TLS and IPSec protocols. To the best of our knowledge, VPN configuration automation has been scarcely studied so far, and only in the context of the IPSec protocol. Moreover, even in this context, no solution is available that solves the problem of automatic CPSs configuration considering all these features together (i.e., automatic computation of both CPS allocation scheme and CPS configuration rules, formal correctness guarantee, and optimization). Therefore, the proposed approach represents a significant step ahead in literature.

In order to achieve our goal, we define a formal model that captures all the information required for communication protection, and we use it for the definition of the configuration problem as a *Maximum Satisfiability Modulo Theories* (MaxSMT) problem. This formulation also lets us introduce some optimization goals that will drive the MaxSMT solver in finding the best solution. The main challenge to address when adopting this approach is to keep the formal model and the corresponding MaxSMT formulation simple enough to obtain a refinement process that is efficient in practice. In order to test the practical applicability and efficiency of the proposed method, we also present a proof-of-concept implementation, and we evaluate it experimentally.

The rest of this paper is structured as follows. Section II dissects the related work to show the novelty of the proposed methodology. Section III provides a high-level overview of the approach that we propose for automating CPSs configuration. Sections IV and V present the formal models defined respectively for network components and communication protection policies and systems. Section VI shows how these models are used for the formulation of the MaxSMT problem. Section VII describes the implementation of the proposed approach and its validation. Finally, Section VIII concludes the paper and discusses future work.

II. Related Work

The automation of network security configuration is widely covered in literature [5]. Several solutions have been proposed that try to minimize, if not completely remove, human interventions in establishing the configuration of network security functions, through automated processes such as policy refinement [6].

However, most of these studies do not target VPNs. Instead, they mainly focus on packet filtering firewalls, which can enforce connectivity policies specifying what traffic flows must be blocked or allowed in a network. The automatic configuration of packet filtering firewalls has been studied in many papers, some also including the use of formal methods [7]–[17]. Even though all these ideas are interesting and useful for automating security configuration, they cannot be directly applied to VPN configuration because CPSs are quite different from firewalls. In particular, filtering operations performed by firewalls only impact the forwarding behavior, whereas encryption and authentication mechanisms employed by VPN gateways may modify the traffic. Besides, deciding the allocation scheme of VPN gateways also requires an analysis of the trustworthiness of the neighboring network nodes, so it is a more complex operation.

The few studies that cover automatic configuration of communication protection via VPNs provide just a few of the features we aim at. Originally, four automated algorithms were proposed for the creation of IPSec-based VPN tunnels, in intra-administration domain scenarios. The first three ones were proposed in [18]: 1) in the "direct" approach, a VPN tunnel is generated to enforce a single policy; 2) in the "bundle" approach, there is an attempt at creating a single tunnel for multiple flows, but the approach does not guarantee the solution optimality; 3) the third approach, called "combined", is a simple combination of the previous ones, to achieve a trade-off between completeness and speed. The fourth algorithm [19], based on the "ordered-split" approach, pursues optimization, aiming to minimize the number of tunnels required to enforce all the policies. All these approaches consider simplified network topologies, i.e. chains where only two CPSs, in known positions, have to be configured. Moreover, they do not use formal models at all.

From these initial ideas, some improvements were made later in a second group of studies [20]–[24]. Specifically, [20] extends the previously proposed approaches to inter-domain environments, where VPN tunnels cross multiple Autonomous Systems, defining a negotiation protocol, through which each Autonomous System can negotiate the VPN configurations with the others. Both [21] and [22] aim to reduce further the number of tunnels required to enforce the requested policies. The former uses an iterative algorithm to remove overlapping or redundancy anomalies among the VPN rules and consequently to reduce their total number, while the latter employs recursive binary trees to reuse already generated VPN rules to satisfy new requirements by only adjusting their selectors, thus avoiding the creation of additional specific rules. Instead, [23] designs an algorithm that provides higher robustness against potential failures and the agility needed by mobile IPsec devices. Finally, [24] introduces support for hybrid SDN architectures, where multiple network devices belonging to different providers should be connected through VPN tunnels.

A third group of studies combines automatic CPSs configuration with automatic configuration of other function types (e.g., firewalls, and intrusion detection systems) [25]–[27], even though their description for the CPSs auto-configuration problem is not as sufficiently detailed as for the other security functions.

The studies of such second and third groups still have

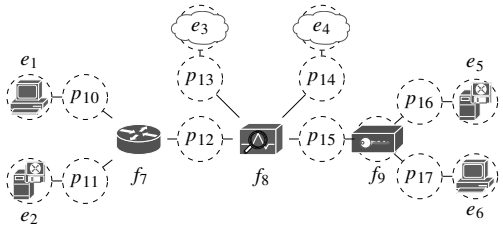


Fig. 1: Example of Allocation Graph

strong limitations with respect to our approach. First, most of them still consider only chain topologies without using formal methods. The only exceptions are [20] and [26]. In particular, [20] considers more general network topologies, but with a fixed set of already positioned CPSs and no formal model at all, while [26] addresses only the allocation problem, without automatic generation of CPS rules. Although [26] uses a formal approach, it is based on an iterative SMT formulation, which is quite different from our MaxSMT formulation. Besides, these studies do not model some aspects, such as how traffic is forwarded in the network, that are necessary to define a correct positioning and configuration of CPSs in the network. Finally, all the existing approaches mentioned above only support IPSec and not TLS, while TLS is becoming another common VPN technology.

From this analysis, it follows that our proposal is the first one that addresses the problem of automatically defining both the allocation scheme and the protection rules of CPSs in a computer network topology, i.e., not necessarily a chain, starting from a set of communication protection policies. Even restricting attention only to the case of chains, our solution is the first one combining automation, formal verification and optimization to solve the CPS allocation and configuration problem.

III. Approach

This section presents the approach proposed in this paper to automatically compute the allocation scheme and protection rules of distributed CPSs.

Specifically, Subsection III-A describes the inputs a user should provide to the proposed approach, whereas Subsection III-B describes the methodology followed for the computation of the output CPS allocation scheme and configuration. In both cases, our objective is to discuss the main ideas behind our proposal at a high level, independently of the actual models and mathematical problem constraints. This description may thus be helpful in understanding the formalization of the inputs and of the methodology presented in next sections.

A. Inputs

The proposed methodology requires three inputs for CPS configuration: (i) an *Allocation Graph* (AG); (ii) a set of *Communication Protection Policies* (CPPs); (iii) an optimization profile.

Allocation Graph

The first input is a graph representing the physical topology of the computer network for which CPSs must be configured. An example of AG is shown in Fig. 1. An AG includes different types of nodes, with different functionalities and roles for communication protection. Each node can have associated configuration parameters (e.g. IP addresses). End points may be single devices (e_1, e_2, e_5, e_6 in Fig. 1) or subnetworks (e_3, e_4 in Fig. 1), and they may contribute or not to the generation of VPNs, according to whether they support the required VPN protocol and cipher suites to generate them (the supported features are configuration parameters associated with each node).

The AG includes special nodes (p_{10}, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}, p_{17} in Fig. 1), called Allocation Places (AP), representing placeholder positions where CPSs may be allocated. These nodes may be present or absent in the output allocation scheme, and their configuration parameters (i.e. their rules) are not yet specified in the AG. Some VPN gateways (f_9 in Fig. 1) may already be present in the AG, and they may be employed for the creation of VPNs, jointly with other CPSs hosted by end points or allocated by the configuration procedure. These gateways are in fixed positions, and they are already part of the allocation scheme to be computed, but their protection rules are not yet specified.

All the other nodes (f_7, f_8 in Fig. 1) that are not end points, APs, or VPN gateways, represent service functions performing other types of operations, such as firewalling, network address translation and load balancing. As the methodology focuses on the configuration of CPSs only, these functions are assumed to be already configured (e.g., firewalls already have their own filtering rules). Their behavior and configuration must be taken into account, as they may impact the resolution of the auto-configuration problem. For example, the filtering rules of a firewall establish what traffic flows are blocked or allowed in a network. Therefore, VPNs must be configured so as to ensure that the traffic crossing them is not dropped by firewalls. Besides, some middleboxes can also support VPN capabilities (e.g., they can support some encryption algorithms) and they can also be considered part of the CPS allocation scheme.

Communication Protection Policies

The second input is a set of CPPs, specifying the communication protection properties that must be enforced in the AG. An example of CPP set is shown in TABLE I. CPPs are expressed with a medium-level language [27], which is at the same time user-friendly and implementation agnostic. We assume that the CPPs, defined by the security administrator, do not overlap with or contradict each other, i.e., they are disjoint. About this aspect, several approaches exist in literature [1], [2], [28] to avoid such anomalies in the policy definition. Moreover, in literature there are also methods to define policies with higher abstractions levels, and to translate them into medium-level languages [29].

Each CPP is characterized by the following pieces of information: 1) policy conditions, used to identify the communications the policy refers to (column 1 of TABLE I); 2) information about how confidentiality and integrity must be enforced for the identified communications (columns 2, 3 and 4 of TABLE I); 3) information about VPN protocols that must be used (column 5 of TABLE I); 4) information about the VPN enforcement modes (column 6 of TABLE I); 5)

TABLE I: *Examples of Communication Protection Policies*

Policy conditions	Algorithms for confidentiality	Algorithms for integrity	Algorithms for authentication	VPN protocols	Enforcement modes	Trustworthiness and Inspection
IPSrc = 192.174.1.∗, IPDst = 144.14.2.∗, pSrc = ∗, pDst = 80, tProto = ∗	{3DES-CBC}	⋆	{NULL}	{TLS, IPSec}	(false, true, d.c., true, d.c., false, false)	N^U = {125.22.2.2}, N^I = {128.28.8.5}, L^T = ∗
IPSrc = 122.33.33.3, IPDst = 12.67.84.2, pSrc = 110, pDst = ∗, tProto = ∗	{AES-GCM-256, 3DES-CBC}	{HMAC-SHA-512}	{RSA}	{TLS}	(true, d.c., true, true, d.c., true, false)	N^T = ∗, L^U = {link between 55.44.33.22 and 55.44.33.27}

information about the trustworthiness and inspection needs of AG nodes and links for the identified communication (column 7 of TABLE I). By communication we mean a flow of packets intended to go from a source node to a destination node, crossing a number of intermediate nodes that may possibly drop them (e.g. firewalls) or modify them (e.g. NATs).

1) *Policy conditions:* They are expressed as predicates on the fields of the IP 5-tuple[1]. The conditions on source IP address and port (IPSrc, pSrc) identify the source of the communication, and refer to the packets sent by the source, while the conditions on destination IP address, port and transport-layer protocol (IPDst, pDst, tProto) identify the destination of the communication and refer to the packets received by the destination. The symbol ∗ can be associated to each field of the IP 5-tuple to specify that all the possibles values must be considered for that packet field for the identification of the communications which are subject to the policy.

2) *Information about how confidentiality and integrity must be enforced*: The way confidentiality and integrity must be enforced is specified by three sets of algorithms (for confidentiality, integrity and client/server authentication, respectively) that are acceptable. In case a security property is not wanted (e.g., a communication should not be encrypted for confidentiality), the only element composing the algorithm set for that property is the NULL algorithm. Besides, the user may decide not to request a specific algorithm. In that case, the special symbol ⋆ is used to indicate that any algorithm can be employed for the security property enforcement.

3) *Information about VPN protocols*: Each policy may require the use of a specific VPN protocol for its enforcement. In particular, all the models discussed in this paper have been defined to be compliant with the two main VPN protocols, i.e., TLS and IPSec. As for the algorithms, the user may decide not to request a specific VPN protocol. The special symbol ⋆ is then used to indicate that any protocol can be used. In that case, the protocol will be actually chosen depending on the available algorithms.

4) *Information about the VPN enforcement modes*: This is expressed through seven Boolean values, specifying the following choices: 1) whether confidentiality must be enforced on the header of the original packet, 2) whether integrity must be enforced on the header of the original packet, 3) whether confidentiality must be enforced on the packet payload, 4) whether integrity must be enforced on the packet payload, 5) whether integrity must be enforced on the header of the encapsulating packet (if any), 6) whether server authentication is requested, and 7) whether client authentication is requested. These seven values are joined into a tuple, and they can take ternary values: true, false, don't care (d.c.). The d.c. value means that the specific enforcement mode is irrelevant for the user.

5) *Information about the trustworthiness of AG nodes and links:* The user must specify in which network sections the communication protection must be enforced, and in which ones it is not required. In particular, some AG nodes are defined "untrustworthy" (N^U) for a policy when every packet belonging to the communication identified by that policy and crossing those nodes must be strengthened with the requested security properties. VPN gateways and APs can themselves be untrustworthy for a policy. In that case, they cannot be used to enforce security properties for the communications identified by the policy. Instead, the AG nodes that are defined "trustworthy" (N^T) for a policy are nodes which the communication packets can cross in plain or strengthened with the requested security properties. The user must specify either the N^U set of untrustworthy nodes or the N^T set of trustworthy nodes, because the other set can be automatically derived by our approach by using the AG topological information. Similar definitions apply to "untrustworthy" links (L^U) and "trustworthy" links (L^T), and again the user must specify only one of these two sets. Moreover, for what concerns inspection needs, some AG node are defined "inspector" nodes (N^I) for a policy if it is required that all the packets belonging to the communication identified by that policy cross those nodes without encryption. The reason is that the node should be allowed to inspect those packets (e.g., because it is an intrusion prevention system or the inspection is required to avoid disclosure of business information). In view of this definition, inspector nodes must also be trustworthy nodes. The specification of the N^I set is optional for the user. Finally, for each one of these sets related to trustworthiness and inspection, the special set denoted by ∗ represents all the possible elements that may be included in those sets. Note that the trustworthiness information is per-policy, e.g., different CPPs might define different untrustworthy nodes (for different

[1]For simplicity, we identify communications on the basis of the IP 5-tuple, but, without loss of generality, other fields (e.g., MAC addresses or URL) can be easily introduced without significant impact on the models that are later defined.

communications).

Optimization profile

The third input is an optimization profile, specifying the criterion to be used for optimization. For this parameter, different choices may be offered to the user. Here, for simplicity we consider just the following two possible profiles, both referred to network efficiency[2].

The *min-allocation* profile aims to allocate the least number of VPN gateways. This achievement would be useful in both traditional and virtualized networks. In the former, less middleboxes would be bought and manually installed. In the latter, less resources of the general-purpose servers would be used for deploying virtual functions. This objective can be achieved by promoting the generation of end-to-end VPNs where end points make their communications secure by themselves, instead of site-to-site VPNs. The counterbalance of optimizing allocation is that in end-to-end VPNs the bandwidth consumption is higher, as the communications that are identified by the security policies are reinforced with the requested properties for their whole paths, even if it was not strictly necessary.

The *min-bandwidth* profile aims to improve the performance of network communications in terms of bandwidth. This objective consists in generating an allocation scheme where VPN gateways are preferred than end points for covering the role of CPSs. This preference causes protection to be enforced for as short as possible paths, therefore saving bandwidth. The counterbalance of optimizing bandwidth is that resource consumption may result higher, as it may be necessary to install a higher number of middleboxes for VPN generation than what would be strictly necessary.

As it is evident, the two profiles have opposite optimization criteria, and they cannot be enforced together.

These two optimization profiles are just demonstrative examples for showing the capabilities of the methodology. On the one hand, if we consider scenarios where optimization goals are not clearly defined or are highly variable, a user may decide to use no optimization profile at all, because an optimization profile is not mandatory for the effectiveness of the overall methodology, or simply select the one of the two that is expected to fit best. It is important to note that even a human network administrator would not be able to keep up with the variability of those scenarios and, consequently, would not be able to find optimized solutions effectively and quickly. Finally, we remark that the proposed approach is flexible enough to support other optimization profiles, which the users may personally define without the need of modifying the overall configuration methodology. If new optimization goals can be quantified, the user may consider them, or a combination of them, in the definition of a new profile[3].

[2]The motivation of this choice is that it would be debatable to define profiles that optimize security parameters, as it is difficult to quantify and estimate the contribution that each VPN solution provides to the security of a computer network.

[3]Even if quantifying the security contribution of VPN solutions is difficult and open to multiple interpretations, if the users can actually quantify them in specific scenarios related to their networks, they may introduce new profiles focused on them, and they would be supported by our methodology.

B. Methodology

The core of the methodology consists of formulating the auto-configuration problem as a MaxSMT problem, and solving it by means of an off-the-shelf solver. A MaxSMT problem enriches and extends a traditional SAT problem, which consists of determining if there exists an interpretation of a set of Boolean formulas that satisfies them all, from two points of view. On one side, while in a SAT problem only Boolean formulas are considered, in a MaxSMT problem logical formulas based on a variety of theories (e.g., integers, strings, bit arrays) can be used, so enabling the modeling of complex problems in a way that is closer to reality. On the other side, a SAT problem is only composed of hard constraints, i.e., a valid interpretation must necessarily satisfy all formulas. Instead, a MaxSMT problem can have not only hard constraints, but also weighted soft constraints. The latter are assigned with weights, and the goal is to maximize the sum of the weights that are assigned to the satisfied soft constrains. Therefore, there is not a strict requirement of satisfaction for these clauses. Soft constraints enable the achievement of optimization objectives. Instead, hard constraints enable the achievement of formal correctness by construction, provided that the properties that must be formally guaranteed in the solution are modeled as hard constraints. For these reasons, formulating our problem as a MaxSMT problem lets us achieve all the three stated objectives, i.e. full automation, optimization, and formal correctness.

The MaxSMT formulation has been chosen also because the resolution of MaxSMT problems has a good scalability on average, and efficient state-of-the-art solvers such as [30] exist. It is indeed necessary to accurately tune the definition of the formulas on which a MaxSMT problem leans, as the way they are defined may impact both the possibility to find solutions and the performance of the problem resolution significantly. This is the main challenge that we had to address to develop our solution using this technique.

According to this approach, we define formal models, based on first-order logic formulas, to model the network components in the AG, and for the CPPs and the CPSs. Such models (respectively described in Sections IV and V) capture all the information that must be considered for the computation of the CPSs configuration. Then, the MaxSMT problem is formulated, by defining both hard and soft constraints (illustrated in Section VI) based on the formal models, and the MaxSMT solver is executed for solving the resulting problem instance. In case at least a hard constraint cannot be fulfilled (i.e., the CPPs are not all feasible), then a non-enforceability report is produced to inform the user that the auto-configuration problem has no solution. Otherwise, the output is composed of the CPSs allocation scheme (i.e., it shows on which APs and other nodes a CPS must be allocated), and the protection rules for each allocated CPS. This information will be exploited by the user to create and configure the security service.

IV. Network Model

This section presents the formal model of the relevant network components: the AG model in Subsection IV-A, the

TABLE II: *Notation*

Symbol	Definition
$\mathbb{B} = \{\text{true, false}\}$	Boolean set
$G = (N, L)$	Allocation Graph (AG)
N	node set of the AG
N^E	endpoints
N^V	VPN gateways
N^A	Allocation Places (APs)
N^F	other middleboxes
$n_k \in N$	the element of N identified by k
$n_s, n_d \in N^A$	source/destination endpoint
L	edge set of the AG
$l_{i,j} \in L$	the edge from n_i to n_j
$t = (p^i, h^i, h^a)$	a class of packets
t^0	the empty set of packets
$t_{i,j}$	the traffic transmitted from n_i to n_j
h^x	a 5-tuple header
p^i	the internal payload
h^i	the original initial header
h^a	the additional header
F	flow set
f	a flow, i.e., class of packets
P	set of CPPs
p	a CPP
C	condition set of a CPP
A^c, A^i, A^a	cipher algorithms accepted by a CPP
V	VPN protocols accepted by a CPP
S	information about CPP security properties
s_{ci}	information about the confidentiality of $t.h^i$
s_{ii}	information about the integrity of $t.h^i$
s_{cip}	information about the confidentiality of $t.p^i$
s_{iip}	information about the integrity of $t.p^i$
s_{ia}	information about the integrity of $t.h^a$
s_{sa}	information about server authentication
s_{ca}	information about client authentication
W	information about network trustworthiness
$N^U \subseteq N$	untrustworthy nodes
$N^T \subseteq N$	trustworthy nodes
$N^I \subseteq N$	inspector nodes
$L^U \subseteq L$	untrustworthy links
$L^T \subseteq N$	trustworthy links
$F^p \subseteq F$	flows that satisfy the the conditions of CPP p
R_n	set of all the placeholder rules of n
r	a placeholder rule
C	condition set of a CPS rule
a^c, a^i, a^a	cipher algorithms enforced by the CPS rule
c	VPN protocol used by the CPS rule
S	information about the security properties enforced by the CPS rule
m	Boolean value that expresses if the traffic satisfying C must be encapsulated through a tunnel
act	action of the CPS rule
$index^N$: $N \to \mathbb{N}_0$	maps $n \in N$ to its non-negative integer index
$address$: $N \to 2^I$	maps $n \in N$ to its set of IP addresses
η: $T \to H$	maps a packet class t to its most external header h
π: $F \to (N)^*$	maps a flow to the ordered list of nodes that are crossed by that flow
τ: $F \times N \to T$	maps a flow and a node to the ingress traffic
ν: $N \times F \to N_A + \{n_0\}$	maps a network node n and a flow f to the next node crossed by f after n
$transform$: $N \to T$	maps a node and an input traffic to the corresponding output traffic
ϕ: $P \to \mathcal{P}(F^P)$	maps a policy to the set of traffic flows computed for it
ρ: $F^P \to P$	maps a traffic flow to the policy for which it has been computed
$allocated$: $N \to \mathbb{B}$	true $\Leftrightarrow$ a CPS is allocated in $n \in N$
$tunneled$: $T \to \mathbb{B}$	true $\Leftrightarrow$ the traffic t has an external header that has been added by a CPS
$deny$: $N \to T \to \mathbb{B}$	true $\Leftrightarrow$ n drops all the packets in t
$protect^x$: $N \times F \to \mathbb{B}$	true $\Leftrightarrow$ a CPS allocated on n adds a protection to $\tau(f, n)$ for the corresponding security property (confidentiality when $x = c$, integrity when $x = i$), or if it starts an authentication procedure (when $x = a$)
$unprotect^x$: $N \times F \to \mathbb{B}$	true $\Leftrightarrow$ a CPS allocated on n removes a protection from $\tau(f, n)$ for the corresponding security property (confidentiality when $x = c$, integrity when $x = i$), or if it completes an authentication procedure (when $x = a$)
$supported$: $N \times \mathbb{A}^c \times \mathbb{A}^i \times \mathbb{A}^a \to \mathbb{B}$	true $\Leftrightarrow$ the three cypher algorithms are supported by the node
$delimiters$: $N \times N \times F \to \mathbb{B}$	true $\Leftrightarrow$ the two nodes delimit a VPN for the flow
$match$: $R_n^{\text{filt}} \times T \to \mathbb{B}$	true $\Leftrightarrow$ the rule conditions match the traffic
$configured$: $R_n \to \mathbb{B}$	true $\Leftrightarrow$ the placeholder rule is configured
$t_1 \subseteq t_2 \in T$	t_1 is a sub-traffic of t_2
$\wedge$, $\vee$, $\neg$	used for conjunction, disjunction, negation
$.$	used to denote a specific tuple element (e.g., given a tuple $t = (a, b, c)$, $t.a$ identifies element a of tuple t)

traffic flows model in Subsection IV-B and the network functions model in IV-C. The network model is partially mutated from the modeling approach described in [12]. However, some changes were introduced to make it compliant with some exclusive characteristics of CPPs, as that modeling approach was designed to work with packet filtering firewalls.

TABLE II includes the main formal notation (symbols, functions, predicates, operators) used in this section, and in all the next sections related to models. Many components are modeled as tuples, composed of sub-components. The "." operator will be used to access a single named sub-component of a tuple in the remainder of this paper. For instance, for a tuple $t = (a, b, c)$, the notation $t.a$ denotes the a sub-component of t.

A. Allocation Graph model

The formal model for the AG is a directed graph, represented by the tuple $G = (N, L)$.

On one side, N is the set of vertices representing the network nodes, and it is composed of four disjoint subsets, i.e., $N = N^E \cup N^V \cup N^A \cup N^F$. Subset N^E includes all the end points which can be the source or destination of a network communication (they may be single devices or full subnetworks, e.g., a branch of a network company). N^V includes all the VPN gateways that are already present in the network topology, whereas N^A is the set of all the APs where new CPSs may be allocated. Instead, N^F consists of all the other network functions (e.g., load balancers, network address translators, packet filtering firewalls) that are present in the network. Some of them may be enabled to host CPS functionalities as well.

Two utility functions are defined on the N set:

- the $index^N$: $N \rightarrow \mathbb{N}_0$ function maps each node to a unique non-negative integer number;
- the *address*: $N \rightarrow 2^I$ function maps each node to the set of its IP addresses.

Instead, the *allocated*: $N \rightarrow \mathbb{B}$ predicate maps a node $n \in N$ to true if n has a CPS capability allocated in the solution (of course, for each VPN gateway $n \in N^V$, which is already present in the AG, *allocated(n)* = true).

On the other side, L is the set of links, i.e. directed arcs interconnecting the nodes of G. Each link is uniquely identified by two non-negative integers, respectively the indexes of the source and destination nodes. Each link is mapped to these integers through the $index^L$: $L \rightarrow \mathbb{N}_0^2$ function.

B. Traffic flows model

A traffic t represents a packet class and it is defined as a tuple $t = (p^i, h^i, h^a)$, where p^i models the internal payload, h^i models the original initial header, while h^a models the additional header that may be added by a VPN[4]. Each h^x, with $x = \{i, a\}$, is a conjunction of five predicates, one for each field of the IP 5-tuple. For simplicity, we write h^x as a tuple, composed of the five predicates, i.e., $h^x = (IPSrc, IPDst, pSrc, pDst, tProto)$.

IPSrc and *IPDst* are predicates that express conditions on the source IP address and on the destination IP address, respectively. These conditions may impose a specific value for the IP address, or specify that it can take a range of values. Each predicate is defined over four integer variables, each one representing a byte of the corresponding IP address. For example, the predicate imposing that the source IP address is equal to 127.0.10.1 is "$x_1 = 127 \wedge x_2 = 0 \wedge x_3 = 10 \wedge x_4 = 1$", where each x_i, with $i \in \{1, 2, 3, 4\}$, is an integer variable that can take values from 0 to 255. For simplicity, we use the following sugared syntax for expressing this predicate in a more concise way: "*IPSrc* = 127.0.10.1". Similarly, "$x_1 = 127 \wedge x_2 = 0 \wedge x_3 = 10 \wedge x_4 >= 0 \wedge x_4 <= 255$" can be concisely written as "*IPSrc* = 127.0.10.$*$".

pSrc and *pDst* are predicates that express conditions on the source transport-layer port and on the destination transport-layer port, respectively. As for the predicates on the IP addresses, they can define a single value or a range of values. However, they are defined over a single integer variable. For example, a possible *pSrc* predicate is "$x = 80$", with x an integer variable, and the corresponding sugared syntax is "*pSrc* = 80". Finally, *tProto* expresses a condition about which transport-layer protocol is used. This predicate is defined over a single variable too. A possible example is "x = TCP", whose corresponding sugared syntax is "*tProto* = TCP".

Each predicate p out of these five ones can have a special formulation, i.e., "$p = *$". This means that any possible value is acceptable and the predicate is always true (e.g., "*IPSrc* = $*$" means that this predicate returns true for any possible value of the four variables on which it is defined).

Denoting the set of all the possible traffics with T, and the set of all the possible 5-tuple-based headers with H, the η: $T \rightarrow H$ function maps a packet class t to its most external header h, among the headers that really exist in the real traffic represented by t. This function works on $t \in T$ as shown in (1): it maps t to $t.h^i$ if the external 5-tuple-based header is totally absent from the traffic, to $t.h^a$ otherwise.

$$\eta(t) = \begin{cases} t.h^a & \text{if } tunneled(t) = \text{true} \\ t.h^i & \text{otherwise} \end{cases} \quad (1)$$

In this definition, the *tunneled*: $T \rightarrow \mathbb{B}$ predicate maps a traffic t to true if t has an external header that has been added by a CPS.

Another definition useful for our model is the definition of sub-traffic. Considering two packet classes $t_1, t_2 \in T$, we say t_1 is a sub-traffic of t_2, written $t_1 \subseteq t_2$, if t_1 represents a subset of the packets represented by t_2 (i.e., $\eta(t_2) \Rightarrow \eta(t_1)$).

After having defined the traffic model, let us introduce the *traffic flow* model, which is a concept related to what we called informally a communication. Denoting the set of all traffic flows with F, a traffic flow $f \in F$ represents how a specific packet class would be transformed when crossing a list of nodes in N. As such, f is formally represented as a list $f = [n_s, t_{sa}, n_a, t_{ab}, n_b, ..., n_j, t_{jk}, n_k, ..., n_p, t_{pd}, n_d]$, with alternating node and traffic elements. In this model, the traffic t_{sa} represents a set of packets that node n_s may generate and

[4]In the proposed model, we chose to avoid the generation of VPNs based on multiply nested tunnels, because those solutions would have highly degraded performance.

send to n_a, while t_{ab} represents the packets that n_a could send to n_b and that would result from the transformation of the packets in t_{sa} (as a special case, traffic may be forwarded without transformation), and so on. Another property of a flow is that the packets expressed by t_{jk} are all managed in the same way by the receiving node n_k, i.e. the decision taken by n_k on each one of them is the same. In other words, packets that are discriminated by the receiving node belong to different flows. A flow f just represents how packets would be transformed if they were forwarded, not the forwarding decisions, i.e. a flow may be interrupted, meaning its packets cannot reach the intended destination because an intermediate node (e.g. a firewall) blocks them.

Three utility functions are defined over domain F:

- π: $F \rightarrow (N)^*$ maps a flow f to the ordered list of nodes that are crossed by f:

$$\pi([n_s, t_{sa}, n_a, t_{ab}, ..., n_k, t_{kd}, n_d]) = [n_s, n_a, ..., n_k, n_d] \quad (2)$$

  For two nodes n_i and n_j belonging to the same node list, $n_i \prec n_j$ means that n_i precedes n_j in that list. Similarly, $n_i \preceq n_j$ means that n_i precedes n_j, or it is n_j itself.
- τ: $F \times N \rightarrow T$ maps a flow f and a node n to the traffic that precedes n in the definition of f. In case n is not crossed by f, i.e., $n \notin \pi(f)$, $\tau(f, n) = t_0$, where t_0 is the element of T that symbolizes absence of traffic. For example:

$$\tau([n_s, t_{sa}, n_a, t_{ab}, n_b, ..., n_k, t_{kd}, n_d], n_b) = t_{ab} \quad (3)$$

- ν: $F \times N \rightarrow N$ maps a flow f and a node n to the node that follows n in $\pi(f)$. If no node that follows n exists, then $\nu(f, n) = n_0$, where n_0 is the element of N that symbolizes absence of node. For example:

$$\nu([n_s, t_{sa}, n_a, t_{ab}, n_b, ..., n_k, t_{kd}, n_d], n_a) = n_b \quad (4)$$

C. Network functions model

The behavior of a network function is defined as a combination of a forwarding behavior and a transformation behavior.

The forwarding behavior of a network function in $n_k \in N$ establishes if a class of packets t_{jk} belonging to a flow $f = [n_s, t_{sa}, ..., n_j, t_{jk}, n_k, t_{ki}, n_i, ..., n_p, t_{pd}, n_d]$ is discarded by n_k or it is forwarded to the next hop n_i. This behavior is modeled by the *deny*: $N \times T \rightarrow \mathbb{B}$ predicate, which maps a node n_k and a traffic t to true if n_k blocks all the packets expressed by t, to false otherwise.

Instead, the transformation behavior of a network function in $n_k \in N$ establishes how a traffic t_{ki} belonging to a flow $f = [n_s, t_{sa}, ..., n_j, t_{jk}, n_k, t_{ki}, n_i, ..., n_p, t_{pd}, n_d]$ is modified by n_k. This behavior is modeled by the *transform*: $N \times T \rightarrow T$ function, which maps a node n_k and an input traffic t to the traffic that may be produced by n_k as output.

These two behaviors have been modeled separately, because their roles in achieving a solution for the configuration problem are different. The transformation behavior is useful to identify how a packet class changes when crossing a chain of nodes, whereas the forwarding behavior defines whether a traffic flow is stopped at a certain point of its path because of a node that drops the input packet class. Thanks to this separation of duties, it is possible to first compute how traffic is transformed when crossing the network, i.e. the *transform* predicate, and later reason about VPN configuration using this information. The *deny* predicate, instead, cannot be computed in advance, because it may depend on how traffic is modified by CPSs.

V. Communication Protection Models

This section illustrates the models for the CPPs in Subsection V-A, and the models for the CPSs in Subsection V-B.

A. Communication Protection Policies model

Let P be the set of the CPPs that must be enforced on AG G_A. Each $p \in P$ is modeled as a tuple $p = (C, A^c, A^i, A^a, V, S, W)$.

C is the condition set, which identifies the communications for which the policy must be enforced. As C takes the same form as the header model H, it is formalized in the same way, as a conjunction of five predicates on the fields of the IP 5-tuple, written as $C = (IPSrc, IPDst, pSrc, pDst, tProto)$. However, as already mentioned, the predicates $C.IPSrc$ and $C.pSrc$ refer to the traffic generated by flow sources, whereas $C.IPDst$, $C.pDst$ and $C.tProto$ refer to the traffic received by flow destinations.

A^c, A^i and A^a represent the cipher algorithms that may be used to respectively enforce confidentiality, integrity and client/server authentication, among all the possible algorithms represented by the $\mathbb{A}^c$, $\mathbb{A}^i$ and $\mathbb{A}^a$ sets. Each A^x, with $x = \{c, i\}$, is modeled as a set of algorithms $A^x = \{a_1, a_2, ..., a_l\}$ that are acceptable for the communication protection requested by p.

V represents the protocols that can be used for VPN creation. In particular, V is modeled as a set of protocols $V = \{v_1, v_2, ..., v_m\}$. In this study, the protocols that can be introduced in this set are TLS and IPSec. However, the model of V has been defined as general as possible, to allow future extensions to other protocols, such as MPLS. Each VPN protocol v_y is characterized by a set of cipher suites that can be used for protection enforcement, i.e., $v_y.CS = \{cs_1, cs_2, ..., cs_q\}$, where each cipher suite is a tuple composed of three algorithms $cs_z = (a^c, a^i, a^a)$ for confidentiality, integrity and client/server authentication, respectively.

S represents the information about how the security properties must be applied on the traffic. It is modeled as a tuple $S = (s_{ci}, s_{ii}, s_{cip}, s_{iip}, s_{ia}, s_{sa}, s_{ca})$, where each component can be a ternary value: true, false, or d.c. (i.e., "don't care"). For traffic t, s_{ci} states if confidentiality must be enforced on the original internal header $t.h^i$, s_{ii} states if integrity must be enforced on $t.h^i$, s_{cip} states if confidentiality must be enforced on the internal payload $t.p^i$, s_{iip} states if integrity must be enforced on the internal payload $t.p^i$, and s_{ia} states if integrity must be enforced on the additional header $t.h^a$ if present. Instead, s_{sa} states if server authentication must be enforced, and s_{ca} states if client authentication must be enforced.

W represents the information about the trustworthiness of network nodes and links. Specifically, W is a tuple $W = (N^U, N^T, N^I, L^U, L^T)$, where:

- $N^U \subseteq N$ is the set of untrustworthy nodes;
- $N^T \subseteq N$ is the set of trustworthy nodes;
- $N^I \subseteq N^T \subseteq N$ is the set of inspector nodes;
- $L^U \subseteq L$ is the set of untrustworthy links;
- $L^T \subseteq L$ is the set of trustworthy links.

As the user can only specify either N^U or N^T, the other set is automatically derived in view of the knowledge of the N set of the AG. In particular, if the user specifies the N^U set, then N^T is computed as $N^T = N \setminus N^U$. Instead, if the user specifies the N^T set, then N^U is computed as $N^U = N \setminus N^T$. Similar considerations apply to the L^U and L^T sets, because the user can specify only one of them.

Given a policy $p \in P$, it is possible to identify all the flows of the F set for which the security properties requested by p must be enforced. Specifically, a flow $f = [e_s, t_{sa}, ..., t_{kd}, e_d]$ satisfies $p.C$ if the following two conditions are true:

- t_{sa} satisfies predicates $p.C.IPSrc$ and $p.C.pSrc$, i.e.,

$$\eta(t_{sa}) \implies (p.C.IPSrc, *, p.C.pSrc, *, *) \tag{5}$$

- t_{kd} satisfies predicates $p.C.IPDst$, $p.C.pDst$ and $p.C.tProto$, i.e.,

$$\eta(t_{kd}) \implies (*, p.C.IPDst, *, p.C.pDst, p.C.tProto) \tag{6}$$

Moreover, in case a flow f satisfies the conditions of multiple policies, as the input policies are disjoint, f is divided into sub-flows, so that each one of them satisfies the condition of a single policy.

The set of all the flows that satisfy the conditions of an element of P is denoted $F^P \subseteq F$. These flows are identified and computed before the formulation of the MaxSMT problem, and then used for the definition of the hard constraints corresponding to the CPPs. In particular, in our methodology, for each policy specified by the user, multiple traffic flows may be computed. In this way, we can handle the existence of multiple paths and other dynamic traffic aspects introduced by functions like load balancers, NATs, etc. In greater detail, for each policy, all the possible network paths (i.e., sequences of nodes) that may be crossed by the flows related to that policy are identified. For example, if a load balancer is present in the network, all the possible forwarding directions are considered, as it is not possible to know how traffic will be actually forwarded a-priori. Then, for each policy, all the possible related flows are computed, considering all paths and all possible transformations of intermediate functions in those paths. For example, if in a path a NAT is the only function that can modify packets and it can change the source IP address into ten different addresses, then ten different flows may be computed for that path, one for each possible transformation. The wildcard $*$ and the address ranges introduced in the traffic model can help in expressing a set of traffic elements compactly, when it is not possible to know beforehand how the packets will be dynamically modified. In this way, dynamic aspects are taken into account.

After the computation of these flows, two utility functions can be employed to relate policies and flows:

- ϕ: $P \rightarrow \mathcal{P}(F^P)$ maps a policy p to the set of traffic flows computed for p;
- ρ: $F^P \rightarrow P$ maps a traffic flow f to the policy p for which it has been computed.

B. Communication Protection Systems model

In the output produced by the MaxSMT solver, each allocated CPS is characterized by a set of rules, establishing which actions the CPS must perform on which packet classes. In order to request the MaxSMT solver to compute this output, the rules of each possible CPS with all their details are represented by a set of free variables for which the MaxSMT solver must find the actual values. More precisely, a set of such variables is associated to each $n \in N$ where a CPS may be allocated, and the rules represented by these sets are called "placeholder rules", as when the problem is formulated it is not yet known whether they will be used or not.

Denoting the set of the placeholder rules of $n \in N$ with R_n, each rule $r \in R_n$ is modeled as a tuple $r = (C, a^c, a^i, a^a, v, S, m, act)$, , which is an abstract model that can be mapped onto the most common secure VPN solutions, such as Azure VPN Gateway and Strongswan. In particular, $C = (IPSrc, IPDst, pSrc, pDst, tProto)$ expresses the conditions that identify the traffic that rule r refers to, while a^c, a^i and a^a respectively specify the confidentiality, integrity and client/server authentication algorithms[5] that are used to enforce protection properties on the traffic identified by C. v specifies the employed VPN protocol (i.e., IPSec or TLS). S specifies how the security properties must be enforced by the CPS on the same traffic, and it is modeled as the S element of a $p \in P$. m is a Boolean value that expresses if the traffic satisfying C must be encapsulated through a tunnel-based VPN (i.e., when m = true) or if communication protection is enforced without an additional header (i.e., when m = false). Finally, act specifies the action the CPS must apply on the traffic that crosses it, i.e., whether the protection required by the security policies must be applied to the traffic that crosses the CPS (i.e., when $act = protect$), or whether it must be removed from it because not necessary anymore (i.e., when $act = unprotect$).

For each node n, and for each policy p and flow f that crosses n , a placeholder rule denoted $r_{p,f}$ is generated, as formally expressed in (7).

$$\forall p \in P. \forall f \in F^P. (n \in \pi(f) \implies r_{p,f} \in R_n) \tag{7}$$

In this way, after the computation of the flows, the cardinality of each R_n can be determined.

Six main predicates, named $protect^c$, $protect^i$, $unprotect^a$, $unprotect^c$, $unprotect^i$, and $unprotect^a$ are introduced for expressing the security properties enforced by CPSs, where:

- $protect^x : N \times F \rightarrow \mathbb{B}$, with $x \in \{c, i, a\}$, maps a node n and a flow f to true if a CPS allocated on n adds a protection for the corresponding security property (confidentiality when $x = c$, integrity when $x = i$) to $\tau(f, n)$, or if it starts an authentication procedure (when $x = a$);
- $unprotect^x : N \times F \rightarrow \mathbb{B}$, with $x \in \{c, i, a\}$, maps a node n and a flow f to true if a CPS allocated on n removes

[5]The NULL algorithm may be specified for each of them.

a protection for the corresponding security property from $\tau(f,n)$ when $x=c$ or $x=i$, or if the CPS must participate to a requested authentication procedure and complete it when $x=a$.

These six predicates, which are related to the free variables representing the placeholder rules, will be heavily used for the MaxSMT problem formulation. The values taken by such predicates are left free as well in the MaxSMT problem formulation.

Additionally, the predicate *supported*: $N\times\mathbb{A}^c\times\mathbb{A}^i\times\mathbb{A}^a\rightarrow\mathbb{B}$ maps a node in N and a triad of algorithms in $\mathbb{A}^c$, $\mathbb{A}^i$ and $\mathbb{A}^a$ to true if those cipher algorithms are supported by that node, i.e., it has the capabilities required to apply them to network packets. For example, if an end point or network function $n\in N^E\cup N^O$ can enforce communication protection properties with a cipher suite composed of AES-128-CBC, HMAC-SHA-256 and RSA, then *supported*(n, AES-128-CBC, HMAC-SHA-256, RSA) = true. Of course, the NULL algorithm is always supported by any node, as using that algorithm means not enforcing the corresponding communication protection property.

Finally, the *delimiters*: $N\times N\times F\rightarrow\mathbb{B}$ predicate is defined to express when two nodes n_i and n_j delimit a VPN for a flow f in order to enforce policy $\rho(f)$. As expressed by (8), n_i and n_j delimit a VPN for a flow f and policy $\rho(f)$ if they belong to the path $\pi(f)$ crossed by f, n_i precedes n_j, n_i enforces the protection required by $\rho(f)$ while n_j removes it, and in-between them there exists no any other node that enforces or removes protection on f for satisfying $\rho(f)$.

$$\begin{aligned}delimiters(n_i,n_j,f)=n_i\in\pi(f)\wedge n_j\in\pi(f)\wedge n_i\prec n_j\wedge\\ protect^x(n_i,f)\wedge unprotect^x(n_j,f)\wedge(\forall n_k\in\pi(f)|n_i\prec n_k\prec n_j.\\ \neg(protect^x(n_k,f)\vee unprotect^x(n_k,f)))\end{aligned}\tag{8}$$

VI. MaxSMT Problem Formulation

This section presents how the hard and soft constraints have been defined for the formulation of the MaxSMT problem.

A. Constraints on CPPs enforcement

Hard constraints are required for expressing the CPPs enforcement, as all of them must be satisfied to achieve any correct solution.

First, the following hard constraints are formulated to express the requirements deriving from the trustworthiness information specified by each CPP $p\in P$.

i) If a node n_i crossed by a flow f satisfying $p.C$ is classified as untrustworthy, then it is necessary that at least a node n_j preceding n_i in $\pi(f)$ enforces the required protection on $\tau(f,n_j)$, and this protection is not removed by any node n_k in-between n_j and n_i:[6]

$$\begin{aligned}\forall f\in\phi(p).\ \forall n_i\in\pi(f)|\ n_i\in p.W.N^U.\\ \exists n_j\in\pi(f)|\ n_j\prec n_i.(protect^x(n_j,f)\wedge\\ (\forall n_k\in\pi(f)|\ n_j\prec n_k\prec n_i.\neg unprotect^x(n_k,f)))\end{aligned}\tag{9}$$

[6]When a formula appearing in this subsection contains the notation $protect^x$ or $unprotect^x$, it means that the formula is valid when x is equal to c, i, or a.

ii) If a link l_{ab} such that nodes n_a and n_b are crossed by a flow f satisfying $p.C$ is classified as untrustworthy, then it is necessary that at least a node n_j preceding n_b in $\pi(f)$ enforces the required protection on $\tau(f,n_j)$, and this protection is not removed by any node n_k in-between n_j and n_b:

$$\begin{aligned}\forall f\in\phi(p).\ \forall l_{ab}\in p.W.L^U|\ n_a,n_b\in\pi(f).\\ \exists n_j\in\pi(f)|\ n_j\prec n_b.(protect^x(n_j,f)\wedge\\ (\forall n_k\in\pi(f)|\ n_j\prec n_k\prec n_b.\neg unprotect^x(n_k,f)))\end{aligned}\tag{10}$$

iii) If a node n_i crossed by a flow f satisfying $p.C$ is classified as inspector, then it is necessary that f does not have confidentiality protection when crossing n_i. Therefore, in case a node n_j preceding n_i in $\pi(f)$ enforces confidentiality on $\tau(f,n_j)$, this protection must be removed by a node n_k in-between n_j and n_i:

$$\begin{aligned}\forall f\in\phi(p).\ \forall n_i\in\pi(f)|\ n_i\in p.W.N^I.\\ \forall n_j\in\pi(f)|\ n_j\prec n_i.\ (protect^c(n_j,f)\implies\\ (\exists n_k\in\pi(f)|\ n_j\prec n_k\prec n_i.\ unprotect^c(n_k,f)))\end{aligned}\tag{11}$$

Then, the following additional hard constraints are formulated to guarantee that the protected traffic can reach its destination, and that it is finally plain.

i) If a node n_i adds protection to a flow f (e.g., because of the previous constraints related to the trustworthiness of nodes and links), then this protection must be removed by a node n_j following n_i in the path followed by f (node n_j can also be the destination end-point of the communication):

$$\begin{aligned}\forall f\in\phi(p).\ (\exists n_i\in\pi(f).\ protect^x(n_i,f))\implies\\ (\exists n_j\in\pi(f)|n_i\prec n_j.\ unprotect^x(n_j,f))\end{aligned}\tag{12}$$

ii) In order to allow a flow f satisfying $p.C$ to reach its destination, it is necessary that none of the nodes in $\pi(f)$ blocks it:

$$\forall f\in\phi(p).\ \forall n_i\in\pi(f).\ \neg deny(n_i,\tau(f,n))\tag{13}$$

Other hard constraints are finally necessary to express the conditions under which a traffic is tunneled by a CPS. In particular, in this model, if the two CPSs n_i and n_j that delimit a VPN for a flow f are end points, then this means that the traffic is not tunneled. Otherwise, an additional header has to be added, with source and destination IP addresses set to those of the CPSs:

$$\begin{aligned}delimiters(n_i,n_j,f)\wedge n_i\in N^E\wedge n_j\in N^E\implies\\ \forall n_k\in\pi(f)|n_i\prec n_k\preceq n_j.\neg tunneled(\tau(f,n_k))\end{aligned}\tag{14}$$

$$\begin{aligned}delimiters(n_i,n_j,f)\wedge\neg(n_i\in N^E\wedge n_j\in N^E)\implies\\ (\forall n_k\in\pi(f)|n_i\prec n_k\preceq n_j.\ tunneled(\tau(f,n_k)))\wedge\\ \tau(f,\nu(f,n_i)).h^a.IPSrc=address(n_i)\wedge\\ \tau(f,n_j).h^a.IPDst=address(n_j)\end{aligned}\tag{15}$$

As a consequence of all these hard constraints, some values of the *protect*, *unprotect* and *deny* predicates may not be free anymore, but they may be forced to be true or false. Of course, these values must be consistent with all the other hard constraints of the problem (e.g., those related to firewalls' behavior, or other CPSs configuration, expressed in subsection VI-B), otherwise the problem is unsolvable.

B. Constraints on network functions behavior

As we have seen, some of the hard constraints are built upon the *deny* predicate, which models the forwarding behavior of the network functions in the AG. In order to correctly model the forwarding behavior of each function in the AG, specific hard constraints are added for each one of them. These constraints depend on the type of service function and on its configuration parameters. For the sake of brevity, here we show how to define these constraints for just two sample types of functions. The same approach can be extended to any other type of function.

The first example we consider is the type of functions that cannot block any packet. Examples of these functions are load balancers, but also APs, as they have not any filtering capability. For a node $n \in N$ that can never block packets, the *deny*(n,t) predicate is simply set to false, independently of the traffic t it is applied to. The hard clause stating this constraint for node n is:

$$\forall\, p \in P.\, \forall\, f \in \phi(p).\, (n \in \pi(f) \implies \neg deny(n, \tau(f, n))) \tag{16}$$

The second example we consider is a packet filter firewall, i.e., a service function that can actively block packets, according to its configuration rules. For the sake of simplicity, it is possible to assume that the configuration of a firewall $n \in N$ is modeled as the tuple $(d_n^{\text{filt}}, R_n^{\text{filt}})$, where d_n^{filt} is the default action ("allow" or "deny"), while R_n^{filt} is the set of the filtering rules. Similarly, each $r \in R_n^{\text{filt}}$ is defined as $r = (a_r^{\text{filt}}, C_r^{\text{filt}})$, with a_r^{filt} as filtering action (it is the inverse of the respective default action) and C_r^{filt} as set of filtering conditions, modeled with five predicates, one for each IP 5-tuple component. Given this model[7], the hard clause that is defined to constrain the *deny* predicate for a firewall is:

$$\begin{gathered} deny(n,t) \iff (a) \vee (b) \\ (a) := (d_n^{\text{filt}} = deny) \wedge (\nexists r \in R_n^{\text{filt}}.\, match(r,t)) \\ (b) := (d_n^{\text{filt}} = allow) \wedge (\exists r \in R_n^{\text{filt}}.\, match(r,t)) \end{gathered} \tag{17}$$

In words, the firewall n blocks the traffic t in two possible cases. The first is that n works in whitelisting mode, and there is no specific allowing rule matching t. The second is that n works in blacklisting mode, and its configuration has a rule matching and allowing t. In this formalization, the *match* : $R_n^{\text{filt}} \times T \to \mathbb{B}$ predicate is used to check whether the conditions of a given rule match a given traffic, i.e., *match*(r, t) = true if $r.C_r^{\text{filt}} \implies \eta(t)$.

C. Constraints on CPSs allocation and configuration

Hard constraints are also necessary to express the CPSs allocation and configuration decisions for each node $n \in N$.

We remind that the allocation decision for node n is modeled by the *allocated*(n) predicate. This value is left free in the problem formulation, and later decided by the solver at runtime, except for VPN gateways, i.e., nodes composing the N^V set, because they already host a CPS functionality. This is expressed by the following hard constraints:

$$\forall n \in N^V.\, allocated(n) \tag{18}$$

The configuration decision is modeled by the *configured* : $R_n \to \mathbb{B}$ predicate which takes value true for a placeholder rule $r \in R_n$ if r is actually used, false otherwise. If *configured*(r) = true, the actual configuration rule is determined by the values assigned to the free variables modeling r, which are constrained by the enforcement of the CPPs, i.e., by the constraints illustrated in Subsection VI-A.

Those constraints act on the protect and unprotect predicates, but such predicates are bound to the parameters of the placeholder rules by the following additional hard constraints:

$$\begin{gathered} protect^x(n,f) \implies \exists r \in R_n.(configured(r) \wedge \tau(f,n) \subseteq r.C \wedge \\ r.act = protect \wedge r.k \in \rho(f).K \wedge r.a^x \in \rho(f).A^x \wedge r.S = \rho(f).S \wedge \\ r.m = tunneled(\tau(f, \nu(f,n))) \wedge supported(n, r.a^c, r.a^i, r.a^a)) \end{gathered} \tag{19}$$

$$\begin{gathered} unprotect^x(n,f) \implies \exists r \in R_n.(configured(r) \wedge \tau(f,n) \subseteq r.C \wedge \\ r.act = unprotect \wedge r.k \in \rho(f).K \wedge r.a^x \in \rho(f).A^x \wedge r.S = \rho(f).S \wedge \\ r.m = tunneled(\tau(f,n))) \wedge supported(n, r.a^c, r.a^i, r.a^a)) \end{gathered} \tag{20}$$

(19) requires that if a CPS n must enforce a communication protection property x (with $x = c$ for confidentiality, $x = i$ for integrity, and $x = a$ for authentication) on traffic flow f due to the requirements of CPP $\rho(f)$, then it must have a configured communication protection rule with a "protect" action, that can enforce the specified protection on $\tau(f, n)$, and it must also support the configured technology and algorithm. Similarly, (20) requires that if a CPS n must remove a communication protection property x on the traffic flow f due to CPP $\rho(f)$, then it must have a configured communication protection rule with an "unprotect" action, that can remove the specified protection from $\tau(f, n)$, and it must also support the configured technology and algorithm, as shown in (20).

Some additional constraints related to CPSs configuration pertain the selection of the VPN protocol (i.e., TLS or IPSec) to be used for the protection enforcement. For example, the triad of algorithms for confidentiality, integrity and client/server authentication that are configured in a communication protection rule $r \in R_n$ must belong to a cypher suite of the VPN protocol selected for the enforcement of that rule:

$$\forall r \in R_n.\, configured(r) \implies ((r.a^c, r.a^i, r.a^a) \in r.v.CS) \tag{21}$$

Another example is that, supposing that a traffic flow crosses a NAT for which the IPsec NAT-Traversal cannot be enabled, then a hard constraint may be introduced to enforce the use of TLS for the protection of that flow.

Finally, if a node has at least a communication protection rule configured , this implies that in that node a CPS has to be allocated, which is expressed by the following additional hard constraint:

$$(\exists r \in R_n.\, configured(r)) \implies allocated(n) \tag{22}$$

D. Constraints on the optimization profiles

The criteria expressed through an input optimization profile are translated into soft constraints. In what follows, the no-

[7]This model supports firewall types that do not require order among the rules except for the default action (i.e., firewalls that work only in blacklisting or whitelisting mode). Besides, the configuration of a firewall with ordered rules can be always equivalently expressed in our model, with algorithms such as the one described in [31].

tation *Soft(c,w)* denotes a soft constraint, expressing clause c and having weight w.

The *min-allocation* profile requires that the allocation of CPS functionalities on end points is preferred over their allocation on intermediate graph nodes. This criterion is enforced via the following soft constraints:

$$\forall e \in N^E.\ Soft(\neg allocated(e), w_e) \tag{23}$$

$$\forall n \in N \backslash N^E.\ Soft(\neg allocated(n), w_n) \tag{24}$$

The first ones correspond to end points, and each one of them is satisfied if no CPS is allocated in the corresponding end point. The second ones correspond to the other nodes, and each one of them is satisfied if no CPS is allocated in the corresponding node. By properly setting the weights of these constraints, as expressed in (25), we achieve the desired optimization goal.

$$\forall n \in N \backslash N^E. \left(\sum_{e \in N^E} w_e \right) < w_n \tag{25}$$

In fact, setting the weight for (24) greater than the sum of the weights for (23) means that it is preferable to allocate CPS functionalities in all the end points (as long as they can support them) rather than in a single intermediate node.

Instead, the *min-bandwidth* profile is characterized by an opposite optimization criterion. In this case it is preferable to allocate CPS functionalities in intermediate nodes (e.g., in APs) rather than in end points, thus saving on bandwidth as communications cross larger network areas without unneeded protection. This criterion can still be expressed through soft constraints (23) and (24). However, the relationship between the weights w_e and w_n is opposite in this case. It is shown in (26), and it states that it is preferable to allocate CPS functionalities in all intermediate nodes rather than in a single end point.

$$\forall e \in N^E. \left(\sum_{n \in N \backslash N^E} w_n \right) < w_e \tag{26}$$

As mentioned in Subsection III-A, the proposed approach is general enough to be extensible with the support of additional optimization profiles. To guarantee their compatibility with the illustrated formalization of the MaxSMT problem, simple soft constraints similar to the ones shown for the *min-allocation* and *min-bandwidth* profiles can be defined, without impacting the definition of any hard constraint.

E. Solution computation

A MaxSMT solver is fed with all the formulated hard and soft constraints, and it searches for the optimal solution that satisfies all hard constraints. If the solver finds a solution, such solution is expressed by the values the solver assigns to the free variables and predicates. In particular, the two outputs, i.e., allocation scheme and protection rules for the CPSs, can be easily retrieved by those values.

On one side, for each $n \in N$, *allocated(n)* shows if a CPS has been allocated by the solver in that network position or not. Therefore, the allocation scheme is defined by all the output values computed for this predicate. On the other side, for each $n \in N$ such that *allocated(n)* = true, the protection rules that must be configured on the corresponding function are the placeholder rules configured by the solver. i.e., the rules $r \in R_n$ such that $configured(r)$ = true. For each such rule r, the solver has assigned a specific value for each free variable modeling r, e.g., it has determined the rule conditions through $r.C$, and the information about the algorithms to be applied through $r.a^c$, $r.a^i$ and $r.a^a$.

The MaxSMT problem formulated according to the proposed approach is decidable, i.e., for any instance of the problem, if all hard constraints can be satisfied, a MaxSMT solver can find an optimal solution that satisfies them. If, instead, such a solution does not exist, a solver can report the unsolvability of the problem. The decidability of the MaxSMT problem is a consequence of the subsets of theories that we used to formulate it (i.e., the Boolean and integer theories, including only relational operators, without quantifiers). In fact, pursuing the objective of keeping the models as lightweight as possible, we avoided more complex integer theories, like the Peano Arithmetic theory, which includes the multiplication operation but would make the MaxSMT problem undecidable. For what concerns quantifiers, which is another possible source of undecidability, we eliminated them from the formulas by using semantics-preserving transformations based on the enumeration of all the limited possible values the quantified variables can take.

If we assume the MaxSMT solver is correct, if it finds a solution then all hard constraints are formally guaranteed to be satisfied, including the ones that represent all the CPPs. This means we have a formal correctness guarantee without the need to apply other time-consuming a-posteriori formal verification steps. Of course, this correctness result holds provided that the model is a faithful representation of the real problem, i.e., the model represents all the information that may influence the solution correctness in a way that is adherent to reality. In the specific case of the MaxSMT problem defined for automatic VPN configuration, the models of the problem inputs must represent all the characteristics of the AG and the CPPs that are required to find a correct solution. Similarly, all the hard constraints modeling the possible forwarding behavior of network functions and the possible traffic flows must be a correct representation of their actual behavior. The models and constraints that have been presented in this paper are straightforward enough to make their adherence to reality evident. Moreover, the modeling of the forwarding behavior of network functions is based on a well-known approach, already used in literature, for which there are also approaches, such as [32], that can extract a formal model of the forwarding behavior of a virtual function automatically from a behavioral representation expressed in a high-level programming language like Java. By using these approaches, it is possible to get high confidence in the adherence of the models to the real function behavior.

Concerning its computational complexity, the MaxSMT problem is NP-complete, as it is a generalization of the SAT problem, which is NP-complete likewise [33]. Nevertheless, many MaxSMT instances can be solved in polynomial time on average using state-of-the-art solvers, like Z3 [34], thanks to algorithms and strategies that have been included in them

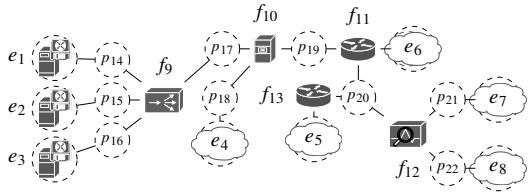


Fig. 2: Allocation Graph of the use case

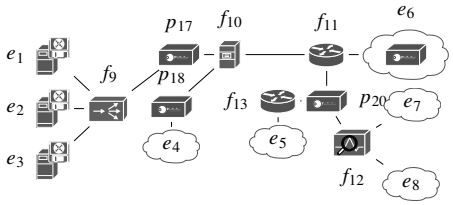


Fig. 3: CPSs allocation scheme of the use case

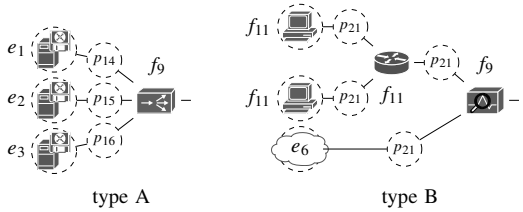


Fig. 4: Sub-graph types for the extensions

to reach the best performance. The actual scalability of these solvers depends on the problem formulation. Keeping the formulation lightweight is a way to improve scalability. Consequently, the MaxSMT problem we defined for automatic VPN configuration has been formulated keeping the number of variables and constraints as low as possible.

It is also worth noting that our modeling approach is independent of the specific MaxSMT solver used for the resolution of the formulated problem. State-of-the-art solvers can include optimizations such as constraint propagation in different ways internally. However, those details are opaque from the perspective of the user in charge of defining the problem to be given as input. Consequently, any solver that adheres to the semantics of MaxSMT problems can be employed without altering the problem formulation or producing different solutions.

VII. Implementation and Validation

A proof-of-concept framework has been developed in Java to prove the feasibility of the proposed approach. This framework adopts Z3 (version 4.8.8) [30], an off-the-shelf MaxSMT solver by Microsoft. A RESTful interface is exposed for interaction with a human being or orchestration tools, which may use the information provided about the CPSs allocation scheme and configuration for deploying the resulting security service. Data exchanged through this interface can be represented in XML or JSON embedding.

The developed framework has undergone manual tests to verify the correctness and optimality of the returned solutions, and scalability tests, aiming to understand the extent to which it can be applied and the benefits it can bring over with respect to an error-prone time-consuming manual configuration. All the tests have been carried out in a 4-core Intel i7-6700 3.40 GHz workstation, equipped with 32 GB RAM.

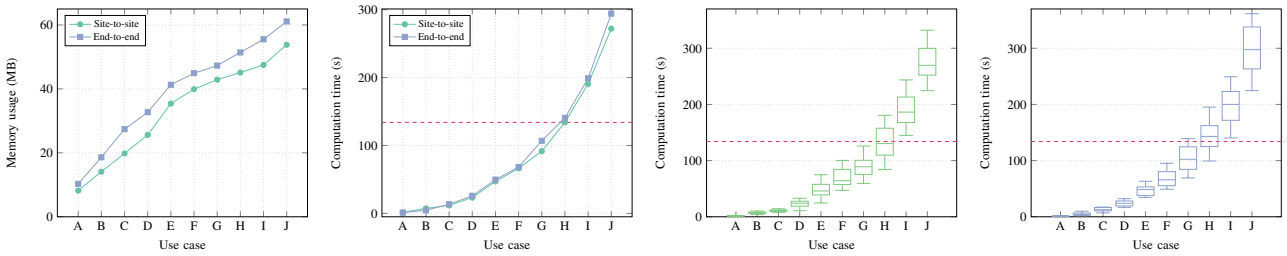
A. Optimization and correctness validation

Even though the solution computed by our methodology is already guaranteed correct and optimal by construction as discussed in Subsection VI-E, we performed some tests to confirm it. Optimization and correctness have been validated with use cases like the one represented in Fig. 2. This picture depicts an AG, inspired by the network topology of our university department. In this scenario, there are different communication protection needs, and due to them, a manual approach would struggle to correctly manage the VPN configuration. For example, the traffic between each research lab (i.e., e_4, e_5, e_6, e_7 and e_8) and the data center, composed of multiple servers (i.e., e_1, e_2, e_3), must be encrypted. Each lab may also require a different encryption algorithm (e.g., AES-GCM-128 for e_4, 3DES-CBC for e_5). Besides, the traffic between each pair of research labs must be protected with integrity guarantees. The only exception is the pair of subnetworks e_7 and e_8, because they are managed by the same lab member. For all these policies, f_{10} and f_{11} are untrustworthy nodes, because no member of the research labs has control over those network functions. Instead, f_{12} is an inspector node, because it is an intrusion detection system that must check all the traffic, according to a binding department rule.

On the basis of these inputs, we have run our framework, and the output topology is shown in Fig. 3. Both the solution optimization and correctness have been checked.

For what concerns optimization, we have enumerated all the possible solutions in configuring VPN gateways for the network depicted in Fig. 2, given the previously described policies. We have thus checked that the output of our framework corresponds to the solution that minimizes the VPN configuration. For example, a single CPS is installed in p_{17} on the right of load balancer f_9, instead of having three separate CPSs on its left. Similarly, a CPS is allocated in p_{20}, so that it can manage communications involving three different subnetworks (e_5, e_7, e_8).

For what concerns correctness, we executed correctness tests in both simulated and real networks. On the one hand, we have simulated the computed VPN configuration with Mininet, a well-known emulator for rapid prototyping and testing virtualized networks. After configuring each element of the network simulated in Mininet, we verified that each communication had the requested communication protection properties by analyzing the traffic on the different emulated nodes. On the other hand, we instantiated the computed VPN configuration in a real virtual environment managed by the container orchestration platform Docker Compose. In this virtual network, each CPS allocated in the solution produced by our methodology is implemented by a containerized Strongswan instance, as Strongswan is one of the most commonly used open-source solutions for VPN creation. The



(a) Memory usage chart (b) Computation time chart (c) Whisker plot (site-to-site) (d) Whisker plot (end-to-end)

Fig. 5: Scalability for increasing problem size

TABLE III: *Characteristics of the use cases for Fig. 5*

	A	B	C	D	E	F	G	H	I	J
$\|P\|$	40	80	120	160	200	240	280	320	360	400
$\|N\|$	35	60	85	110	135	160	185	210	235	260
$\|N^A\|$	10	20	30	40	50	60	70	80	90	100
$\|N^U\|$	5	8	10	13	15	18	20	23	26	30
$\|F^P\|$	40	80	120	160	200	240	280	320	360	400

configuration of each Strongswan instance is easily derived from the communication protection rules computed by our framework, with the support of a simple automatic translator that we have implemented. Indeed, this operation is a mere syntax conversion, not involving policy refinement. Within such a configured environment, we used the *tcpdump* packet analyzer to analyze the structure and content of the packets received by different network nodes. In particular, we verified that packets reaching an untrusted node have the required communication protection properties requested by the corresponding CPP, and we also verified that packets crossing an inspector node are plain so that they can be analyzed. These experimental tests allowed us to check the automatically computed VPN configuration is correct and compliant with the requested CPPs.

Similar experiments have been done with variations of this use case. These variations are obtained by adding a progressively higher number of middleboxes and APs in the backbone of the initial network example, i.e., in the sequence of nodes between f_9 and f_{12} of Fig. 2, and by attaching new sub-graphs to them. Specifically, we used the two sub-graph types depicted in Fig. 4. The sub-graph of type A represents a group of three servers connected to a load balancer through three APs, whereas the sub-graph of type B represents the interconnection of sub-networks with two isolated clients through a ramified topological structure characterized by four APs and two middleboxes. In turn, the right-most middleboxes of the two sub-graph types are used to interconnect recursively other sub-graphs in further extensions of the use cases. In this way, it was possible to create networks of extensive size that were also employed for the scalability tests discussed in the following subsection.

B. Performance and scalability evaluation

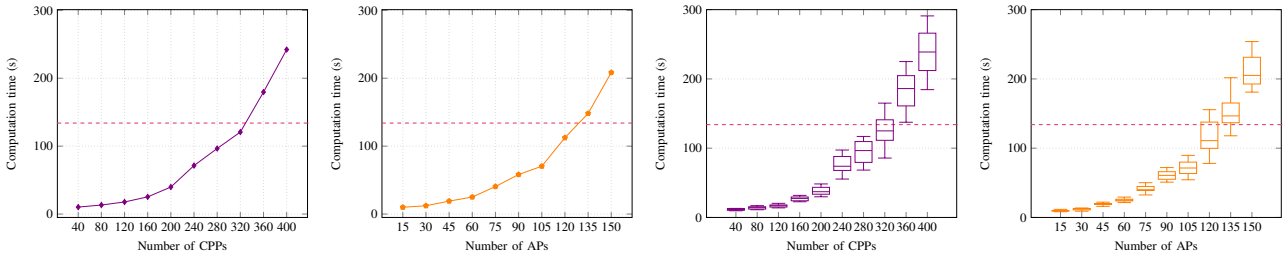
The objectives of performance and scalability evaluation are: 1) to analyze the behavior of the framework in terms of memory usage and computation time for increasing problem size when computing the configuration of different VPN types (i.e., site-to-site and end-to-end VPNs); 2) to assess the impact of the two main parameters (i.e., the number of APs and the number of CPPs) on performance; 3) to compare the performance results in terms of computation time to those of the related state of the art approaches and to the time taken to deploy the computed solutions.

1) Scalability for increasing problem size: Fig. 5 reports the results obtained for scalability versus problem size. These tests have been carried out in ten use cases, represented by network topologies artificially synthesized as extensions of the one depicted in Fig. 2, as previously described in VII-A. TABLE III reports the main characteristics of the ten use cases, in terms of the number of CPPs to be enforced ($|P|$), the number of network nodes ($|N|$), the number of APs where CPSs may be allocated ($|N^A|$), the number of untrusted nodes for each single policy ($|N^U|$), and the number of traffic flows related to the requested CPPs ($|F^P|$). The ten use cases are characterized by increasing values of these parameters, which implies increasing complexity of the MaxSMT problem to be solved. The increase of the values is kept proportional so as to keep a ratio 1:4 between the numbers of APs and CPPs, because, in realistic scenarios, the number of traffic flows to be protected is commonly higher than the number of positions where CPSs can be positioned. Besides, for each use case, the framework behavior has been analyzed for the creation of two different VPN types: site-to-site VPNs if the network endpoints cannot host a VPN capability, and end-to-end VPNs on the contrary.

On the one hand, Fig. 5a shows the peak memory usage of the framework, when the value of all the parameters progressively increases. As it can be seen from the chart, even in the worst case that has been analyzed (i.e., a scenario composed of 260 network nodes and 400 CPPs), the peak memory usage that has been measured is not of concern (61.1 MB).

On the other hand, Fig. 5b shows the computation time of the framework for the generation of site-to-site and end-to-end VPNs. Each plotted value represents the average computed over 50 iterations, where the same network topology and CPPs are kept, and only the IP addresses of the nodes vary. This choice is motivated by the fact that the performance of Z3 is known to vary substantially with the variation of the values of integer constants, and integers are used in the formulation of

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3409073



(a) Computation time chart (increasing CPP number)

(b) Computation time chart (increasing AP number)

(c) Whisker plot (increasing CPP number)

(d) Whisker plot (increasing AP number)

Fig. 6: Scalability for increasing numbers of CPPs and APs

our MaxSMT problem for IP addresses. This time distribution is also confirmed by the whisker plots depicted in Fig. 5c and Fig. 5d, showing the value distribution for the two VPN configuration types.

A first consideration that we can draw from these three charts is that the computation time does not increase exponentially, which allows the framework to complete the computation even on networks with hundreds of nodes and hundreds of CPPs. This result has been achieved thanks to the careful modeling choices we made. Another consideration is that the times for the configuration computation of site-to-site and end-to-end VPNs are comparable. This result shows that our approach can manage multiple VPN types, without relevant differences in performance.

2) Scalability for increasing numbers of CPPs and APs: Fig. 6 reports the results obtained for scalability versus the numbers of CPPs and APs. Again, the network topologies used for these tests are artificially synthesized as extensions of the one depicted in Fig. 2, and 50 runs of the framework are performed for each pair of CPP and AP numbers. The results plotted in Fig. 6a and Fig. 6c are obtained by fixing the AP number to 20 and varying the CPP number from 40 to 400, while those plotted in Fig. 6b and Fig. 6d are achieved by fixing the CPP number to 100 and varying the AP number from 15 to 150.

From the plotted values, it is possible to underline how neither of the two parameters has a drastic impact on the performance when the other one is kept fixed. Of course, looking at the two charts carefully, the ratio between computation time and the number of considered entities (CPPs and APs) is in favor of the number of CPPs, which consequently has less impact than the number of APs. This result is explained by the fact that for each possible node enforcing communication protection a certain number of placeholder rules must be defined, depending on the number of traffic flows related to the CPPs crossing it. Each rule is composed of free variables, and for each one of them the solver must decide the optimal value. Therefore, the number of possible solutions becomes quite high. Nevertheless, if we consider a topology composed of over 200 nodes, establishing how to allocate the CPSs on it and how to establish their rule sets would be an impractical task for a human being, as it would take hours and probably would end up with errors or sub-optimizations.

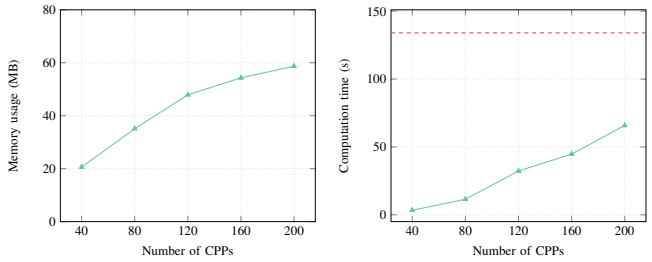
3) Scalability on topologies of real-world networks: The memory and time scalability of the developed framework has also been validated on two AGs inspired by real-world production networks, i.e., APAN[8] and GÉANT[9], respectively located in Asia and Europe. The objective of these additional tests was to assess how the proposed methodology behaves when applied to real-world network topologies characterized by a higher degree of complexity and variability.

TABLE IV reports the main characteristics of the two AGs. In addition to this numerical information, a noteworthy consideration about these AGs is that they have a much more complex and ramified structure than the synthetic topology employed in the previous scalability tests. Consequently, the number of traffic flows that satisfy the conditions of each CPP is higher because there are multiple paths interconnecting the endpoints identified by the CPP itself. The actual numbers of traffic flows, depending on the number of requested CPPs, is shown in TABLE V, to highlight the fact that they are higher than the ones reported in TABLE III. Also, the number of intermediate nodes crossed by each traffic flow is higher. In particular, these characteristics, i.e., the number of traffic flows related to each CPP and the length of their crossed paths, are higher for the GÉANT topology than for the APAN network, because the ramification of the former is more pronounced and produces multiple possible paths to reach any possible destination.

The scalability tests on these topologies were carried out by progressively increasing the number of CPPs that must be enforced on them, measuring how memory and time vary. The results of these tests are plotted in Fig. 7 and Fig. 8.

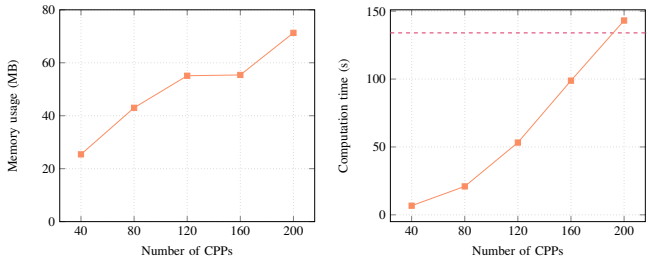
On the one hand, Fig. 7a and Fig. 8a show the peak memory usage of the framework, when applied to the AGs of the two production network AGs. The achieved results are almost the same as the ones obtained for synthetic networks and previously shown in Fig. 5a. From this point of view, the higher complexity of these AGs, represented by the presence of more and longer traffic flows, requires the introduction of some more variables and constraints in the definition of the MaxSMT problem. Nevertheless, as we tried to keep all models as lightweight as possible, considering that the core of the MaxSMT problem remains the same, just a slightly higher amount of memory is required to manage

[8]https://apan.net/
[9]https://network.geant.org/



(a) Memory usage chart (b) Computation time chart

Fig. 7: Scalability on the APAN topology



(a) Memory usage chart (b) Computation time chart

Fig. 8: Scalability on the GÉANT topology

TABLE IV: *Characteristics of the APAN and GÉANT AGs*

	APAN AG	GÉANT AG
Number of vertices	52	49
Number of end points	25	19
Number of APs	19	11
Number of directed links	104	85

TABLE V: *Number of flows in the APAN and GÉANT AGs, depending on the number of CPPs*

		Number of flows	
		APAN AG	GÉANT AG
Number of CPPs	40	61	73
	80	127	150
	120	197	212
	160	265	306
	200	331	379

these two networks, because the impact of each new variable or constraint is limited. This result thus confirms that the developed framework does not require extensive memory to be applied successfully to solve the automatic VPN configuration problem in topologies of real-world networks.

On the other hand, Fig. 7b and Fig. 8b show the average computation time of the framework over 50 iterations, when applied to the AGs of the two production networks. Almost all the execution times are in the same magnitude order as the ones obtained for synthetic networks, shown in Fig. 5b, and the trend of the plots is also the same. The only exception is represented by the time requested by the framework to solve the MaxSMT problem formulated for the enforcement of 200 CPPs on the GÉANT topology. However, even in this case, the increase is only slightly higher, and it is in line with the trend of the plot for that topology. Besides, from these two charts, it is possible to notice that the computation time requested to solve the configuration problem is higher on GÉANT than on APAN. This is explained by the fact that, even though the former is characterized by fewer vertices, it has a more ramified structure, and therefore, the MaxSMT problem to be solved is more complex. This result shows that our methodology can be applied successfully to networks with a complex structure, as the time increase for their management is not significantly high.

In view of all these considerations, the results achieved from these tests show that the developed framework works successfully on topologies of production networks, with memory and time requirements that are in line with the ones requested to manage network topologies artificially synthesized as extensions of a basic model. Therefore, they confirm that the proposed methodology is general enough to support the diverse and complex configurations found in real-world environments.

4) Comparison with state of the art and network deploy times: As previously discussed in Section II, to the best of our knowledge, the approach presented in this paper is the first one in literature to compute both the CPS allocation scheme and the related protection rules jointly, achieving, at the same time, full automation, optimization, and formal correctness. Therefore, a straight performance comparison of our approach to the other existing methodologies would be unfair, or even impossible to some extent, as they either address simpler problems or provide fewer features. Nevertheless, we present here a rough comparison with some of the related state-of-the-art methodologies just to evaluate which impact on performance derived from introducing the additional features of our approach.

Unfortunately, most of the papers proposing the approaches discussed in Section II do not provide an explicit experimental performance evaluation of the proposed approaches. The only ones whose performance has been experimentally evaluated are those described in [18], [27], and [26]. Hence, we can roughly compare the performance figures of our approach only to the ones mentioned in those papers.

The approach described in [18] is specific to VPN configuration, but it cannot compute the CPS allocation scheme automatically. It can only generate the protection rules for CPSs whose position is already fixed in a network chain, without being able to solve the same problem for a ramified topology, and it does not use formal models. In terms of performance, [18] reports that, to compute the CPS rules in a small chain whose size is comparable to the smallest number of APs (i.e., 10) of the networks where our framework has been validated, that approach takes around one hundred of seconds to enforce some hundreds of security requirements. In comparison with that, under similar conditions of input sizes, the execution time of our framework is one magnitude order less, despite addressing both the rule computation and the allocation computation in a ramified graph.

The other two approaches are not specifically designed for VPNs, but include CPSs as possible functions to work with. On the one hand, the technique described in [27] has features

and performance similar to the ones of [18]. In fact, it takes some seconds just to configure a small number of CPSs in a simple chain, without addressing the allocation problem, and without providing any formal correctness for the computed solution. Besides, it does not fully catch the complexity of CPS configurations (e.g., it does not take decisions in terms of VPN protocol or cipher algorithms to be used depending on untrusted and inspector nodes), because the focus of that study is not specifically on CPSs. On the other hand, the approach in [26] only establishes the CPS allocation scheme without computing protection rules. For a small network with around 20 positions where CPS may be allocated, the computation time is in the same magnitude order (i.e., tens of seconds) as with our approach which, however, addresses a more complex problem, as it also computes the CPS rules.

In summary, although the state-of-the-art related approaches address simpler or partial problems, the performance of our methodology is still in line with the state of the art. Of course, also the scalability of our approach has some limitations, as the developed framework cannot manage computer networks composed of thousands of nodes. However, we experimentally showed that it can solve VPN configuration problems of significant sizes, i.e., it can automatically enforce hundreds of CPPs in networks with hundreds of nodes. Therefore, this result can be considered significant by itself.

Finally, we can also compare the time taken by our framework to the time that is commonly required to set up a network. If we consider networks where functions are virtualized, i.e., the ones with the fastest setup times, the computation time of our framework is in the same magnitude order as the Deployment Process Delay (DPD), i.e., the time needed for a state-of-the-art NFV orchestrator, Open Source MANO, to deploy a single virtualized function on a server. The value of this time is 134s according to [35], and it is plotted as a red dashed horizontal line in all figures plotting results about the computation time of our framework. In physical networks, the difference is even bigger, as the manual installation of a middlebox requires much more time.

VIII. Conclusions and Future Work

This paper illustrated a novel approach for the automatic VPN configuration in modern computer networks. The proposed methodology advances the state of the art because, differently from previous solutions, it achieves all the following goals: (i) it computes both the allocation scheme of VPN terminators and their protection rules in a fully automated way, (ii) it is based on a formal model that captures all the relevant aspects of the problem, and provides formal correctness assurance of the solution without requiring a-posteriori formal verification, (iii) it finds the optimal solution, among all the correct ones, according to a user-selected optimization goal.

A Java framework has been developed to implement this approach, and its features (optimization, correctness, scalability) have been validated. From our experiments it results that the proposed approach is viable and scales up to use cases with hundreds of possible positions for VPN gateways and hundreds of protection policies, taking times that, for the most complex tested scenarios, are in the order of hundreds of seconds. This approach is surely convenient from all points of view (correctness assurance and time taken) with respect to a manual configuration, which is notoriously time consuming and error prone.

This result is just one step towards security automation. A possible next future work is to integrate algorithms based on Artificial Intelligence for policy specification, so that our proposed configuration mechanism could work not only on human-specified policies, but also policies automatically derived from network analysis. Additionally, the presented approach could be extended to other types of communication protection policies, so as to support different technologies for generating secure VPNs (e.g., SSH and WS-Security).

References

[1] H. H. Hamed, E. S. Al-Shaer, and W. Marrero, "Modeling and verification of ipsec and VPN security policies," in *13th IEEE International Conference on Network Protocols (ICNP 2005), 6-9 November 2005, Boston, MA, USA*. IEEE Computer Society, 2005, pp. 259–278.

[2] F. Valenza, C. Basile, D. Canavese, and A. Lioy, "Classification and analysis of communication protection policy anomalies," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2601–2614, Oct 2017.

[3] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[4] D. Kreutz, F. M. V. Ramos, P. J. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[5] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, "Automation for network security configuration: State of the art and research trends," *ACM Comput. Surv.*, vol. 56, no. 3, pp. 57:1–57:37, 2024.

[6] R. Boutaba and I. Aib, "Policy-based management: A historical perspective," *J. Netw. Syst. Manag.*, vol. 15, no. 4, pp. 447–480, 2007.

[7] Y. Bartal, A. J. Mayer, K. Nissim, and A. Wool, "*Firmato*: A novel firewall management toolkit," *ACM Trans. Comput. Syst.*, vol. 22, no. 4, pp. 381–420, 2004.

[8] P. Verma and A. Prakash, "FACE: A firewall analysis and configuration engine," in *2005 IEEE/IPSJ International Symposium on Applications and the Internet (SAINT 2005), 31 January - 4 February 2005, Trento, Italy*, 2005, pp. 74–81.

[9] D. Ranathunga, M. Roughan, P. Kernick, and N. Falkner, "The mathematical foundations for mapping policies to network devices," in *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016), Lisbon, Portugal, July 26-28, 2016*, 2016, pp. 197–206.

[10] A. El-Hassany, P. Tsankov, L. Vanbever, and M. T. Vechev, "Netcomplete: Practical network-wide configuration synthesis with autocompletion," in *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*, 2018, pp. 579–594. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/el-hassany

[11] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Automated optimal firewall orchestration and configuration in virtualized networks," in *NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020*. IEEE, 2020, pp. 1–7.

[12] ——, "Automated firewall configuration in virtual networks," *IEEE Trans. Dependable Secur. Comput.*, vol. 20, no. 2, pp. 1559–1576, 2023.

[13] J. Govaerts, A. K. Bandara, and K. Curran, "A formal logic approach to firewall packet filtering analysis and generation," *Artif. Intell. Rev.*, vol. 29, no. 3-4, pp. 223–248, 2008.

[14] P. Bera, S. K. Ghosh, and P. Dasgupta, "Policy based security analysis in enterprise networks: A formal approach," *IEEE Trans. Network and Service Management*, vol. 7, no. 4, pp. 231–243, 2010.

[15] S. Maity, P. Bera, and S. K. Ghosh, "Policy based ACL configuration synthesis in enterprise networks: A formal approach," in *International Symposium on Electronic System Design, ISEDs 2012, Kolkata, India, December 19-22, 2012*, 2012, pp. 314–318.

[16] N. Stouls and M. Potet, "Security policy enforcement through refinement process," in *7th International Conference of B Users, Besançon, France, January 17-19, 2007, Proceedings*, 2007, pp. 216–231.

[17] A. K. Bandara, A. C. Kakas, E. C. Lupu, and A. Russo, "Using argumentation logic for firewall configuration management," in *11th IFIP/IEEE International Symposium on Integrated Network Management, Long Island, NY, USA, June 1-5, 2009*, 2009, pp. 180–187.

[18] Z. Fu and S. F. Wu, "Automatic generation of ipsec/vpn security policies in an intra-domain environment," in *Operations & Management, 12th International Workshop on Distributed Systems, DSOM 2001, Nancy, France, October 15-17, 2001. Proceedings*, 2001, pp. 279–290.

[19] Y. Yang, C. U. Martel, and S. F. Wu, "On building the minimum number of tunnels: an ordered-split approach to manage ipsec/vpn policies," in *IEEE/IFIP Network Operations and Management Symposium, Seoul, Korea, 19-23 April 2004*, 2004, pp. 277–290.

[20] Y. Yang, Z. J. Fu, and S. F. Wu, "BANDS: an inter-domain internet security policy management system for ipsec/vpn," in *IFIP/IEEE Eighth International Symposium on Integrated Network Management (IM 2003), March 24-28, 2003, Colorado Springs, USA*, 2003, pp. 231–244.

[21] C. Chang, Y. Chiu, and C. Lei, "Automatic generation of conflict-free ipsec policies," in *Formal Techniques for Networked and Distributed Systems - FORTE 2005, 25th IFIP WG 6.1 International Conference, Taipei, Taiwan, October 2-5, 2005, Proceedings*, 2005, pp. 233–246.

[22] M. M. G. Sadeghi, B. M. Ali, H. Pedram, M. Dehghan, and M. Sabaei, "A new method for creating efficient security policies in virtual private network," in *Collaborative Computing: Networking, Applications and Worksharing, 4th International Conference, CollaborateCom 2008, Orlando, FL, USA, November 13-16, 2008, Revised Selected Papers*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 10. Springer / ICST, 2008, pp. 663–678.

[23] M. Rossberg, G. Schaefer, and T. Strufe, "Distributed automatic configuration of complex ipsec-infrastructures," *J. Network Syst. Manage.*, vol. 18, no. 3, pp. 300–326, 2010.

[24] L. Firdaouss, A. Bahnasse, B. Manal, and Y. Ikrame, "Automated VPN configuration using devops," in *The 12th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2021) / The 11th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2021), Leuven, Belgium, November 1-4, 2021*, ser. Procedia Computer Science, vol. 198. Elsevier, 2021, pp. 632–637.

[25] J. D. Guttman and A. L. Herzog, "Rigorous automated network security management," *Int. J. Inf. Sec.*, vol. 4, no. 1-2, pp. 29–48, 2005.

[26] M. A. Rahman and E. Al-Shaer, "Automated synthesis of distributed network access controls: A formal framework with refinement," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 2, pp. 416–430, 2017.

[27] C. Basile, F. Valenza, A. Lioy, D. R. Lopez, and A. P. Perales, "Adding support for automatic enforcement of security policies in NFV networks," *IEEE/ACM Trans. Netw.*, vol. 27, no. 2, pp. 707–720, 2019.

[28] C. Basile, D. Canavese, A. Lioy, and F. Valenza, "Inter-technology conflict analysis for communication protection policies," in *Risks and Security of Internet and Systems - 9th International Conference, CRiSIS 2014, Trento, Italy, August 27-29, 2014, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 8924. Springer, 2014, pp. 148–163.

[29] J. Moffett and M. Sloman, "Policy hierarchies for distributed systems management," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 9, pp. 1404–1414, 1993.

[30] L. M. de Moura and N. Bjørner, "Z3: an efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, 2008, pp. 337–340.

[31] H. Hu, G. Ahn, and K. Kulkarni, "Detecting and resolving firewall policy anomalies," *IEEE Trans. Dependable Secur. Comput.*, vol. 9, no. 3, pp. 318–331, 2012.

[32] G. Marchetto, R. Sisto, M. Virgilio, and J. Yusupov, "A framework for user-friendly verification-oriented VNF modeling," in *41st IEEE Annual Computer Software and Applications Conference, COMPSAC 2017, Turin, Italy, July 4-8, 2017. Volume 1*. IEEE Computer Society, 2017, pp. 517–522.

[33] M. E. Halaby, "On the computational complexity of maxsat," *Electron. Colloquium Comput. Complex.*, 2016.

[34] R. Robere, A. Kolokolova, and V. Ganesh, "The proof complexity of SMT solvers," in *Computer Aided Verification*. Springer International Publishing, 2018.

[35] G. M. Yilma, F. Z. Yousaf, V. Sciancalepore, and X. P. Costa, "Benchmarking open source NFV MANO systems: OSM and ONAP," *Comput. Commun.*, vol. 161, pp. 86–98, 2020.

Daniele Bringhenti received the M.Sc. degree (summa cum laude) and the Ph.D. degree (summa cum laude) in computer engineering from the Politecnico di Torino, Torino, Italy, in 2019 and 2022 respectively, where he is currently a Postdoctoral Researcher. His research interests include novel networking technologies, automatic orchestration and configuration of security functions in virtualized networks, formal verification of network security policies.

Riccardo Sisto received the Ph.D. degree in Computer Engineering in 1992, from Politecnico di Torino, Italy. Since 2004, he is Full Professor of Computer Engineering at Politecnico di Torino. His main research interests are in the area of formal methods, applied to distributed software and communication protocol engineering, distributed systems, and computer security. He has authored and co-authored more than 100 scientific papers. He is a Senior Member of the ACM.

Fulvio Valenza received the M.Sc. degree (summa cum laude) and the Ph.D. degree (summa cum laude) in computer engineering from the Politecnico di Torino, Torino, Italy, in 2013 and 2017, respectively, where he is currently a Tenure-Track Assistant Professor. His research activity focuses on network security policies, orchestration and management of network security functions in SDN/NFV-based networks, and threat modeling.