A Formal Model of Security Capabilities towards Vendor-Agnostic Channel Protection

A Formal Model of Security Capabilities towards Vendor-Agnostic Channel Protection

Davide Colaiacomo, Cataldo Basile
Dipartimento di Automatica e Informatica, Politecnico di Torino, Torino, Italy
e-mail: {*name.surname*}@*polito.it*

***Abstract*—This paper presents the Capability Model - Channel Protection (CM-CP), a formal model to abstract the security capabilities of channel protection implementations. Using a Model-Driven Engineering approach, this model forms the basis of a generic policy translator, which converts secure communication policies, written in a vendor-agnostic language, into low-level configurations for specific implementations (known as Security Controls or Network Security Functions). As a result, network administrators can conceive these policies without acknowledging the underlying technologies, thus reducing the potential for errors arising from human intervention. The effectiveness of this work was validated with three state-of-the-art open-source Security Controls: XFRM, StrongSwan, and OpenVPN. As a result, the model's expressiveness and capacity to address concrete requirements for secure channel scenarios are verified.**

***Index Terms*—formal model, security capability, channel protection**

I. Introduction

Establishing secure communication channels is essential for transmitting data while adhering to specific security requirements. However, the increasing sophistication of cyber threats and the diverse scenarios and technologies add pressure on security teams. Thus, human errors undermine the configuration and maintenance of tools designed to counter these threats, making them vulnerable to exploitation by cyber attackers.

Furthermore, integrating new technologies can enhance the effectiveness of defensive measures. However, security teams typically rely on a limited number of vendors to simplify management. This phenomenon, known as vendor lock-in, causes organizations to become heavily dependent on specific technologies, limiting their ability to switch to different solutions without significant resource costs. Vendor lock-in can manifest subtly through time-consuming and less efficient tasks, which is unacceptable in security-critical situations.

The primary objective of this work is to enhance the management of *Security Controls (SCs)* by supporting their configuration with a common high-level language. Consequently, network administrators can define general security requirements, which are then automatically translated into low-level configurations for target technologies. This approach eliminates the need to manage device conventions directly and facilitates the continuous enforcement of these requirements.

To achieve this objective, the main contribution is the Capability Model - Channel Protection (CM-CP), a formal model of the security capabilities, i.e., the features, of the SCs that protect communication channels. With a Model-Driven Engineering (MDE) approach, this model enables the functioning of the Security Capability Manager (SCM), an ecosystem of tools for translating abstract security configurations into vendor-specific ones by leveraging the structure and data within the CM-CP.

The results demonstrate that combining an abstract model with an automatic translation feature is feasible and beneficial. Network administrators can complete traditional manual configuration tasks more quickly and with a lower risk of errors. This allows them to focus on defining channel security properties without concerns about device-specific conventions.

The paper is structured as follows. Section II presents the fundamental concepts about the technologies and the scenarios considered. Section III describes the proposed framework, focusing on how it exploits the model to translate abstract policies into low-level ones. Section V analyses the open issues related to channel protection modeling and the strategies adopted to face them. Section VI presents the validation of the results. Section VII presents previous works. Section VIII provides conclusions and hints for future works.

II. Background

This section presents the protocols and technologies for securing communication that we considered, and the practical scenarios analyzed for validation.

A. Channel protection protocols

Several channel protection protocols are available to enforce communication security. The preliminary study of the current state of the art aimed to cover as many alternatives as possible. To this end, the focus was first on protocols widely used in real infrastructures for secure communications.

IPsec and IKE. The Internet Protocol (IP), the standard for network layer communication, lacks native mechanisms to secure transmitted data. To address this, *Internet Protocol Security (IPsec)* was introduced to protect IP traffic [1]. Complementing IPsec, the *Internet Key Exchange (IKE)* protocol establishes a secure channel between devices by negotiating security algorithms and parameters, known as a Security Association (SA) [2]. SAs enforce the rules defined by a Security Policy (SP), which specifies the processing options for IP traffic. We considered these protocols to cover secure communication channels at the *Internet* layer of the TCP/IP stack, as they are used in many real-world scenarios.

TLS. The Transport Layer Security (TLS) protocol is the standard for securing communications at the transport layer

and was developed to overcome vulnerabilities in traditional protocols, such as Transmission Control Protocol (TCP) [3]. It establishes a secure channel through a handshake, during which cryptographic algorithms and session keys are negotiated. The result is a session state that governs data protection. This protocol was analyzed to cover secure channels built on top of *transport* layer connections.

B. Channel protection implementations

After identifying the protocols for securing communication channels, we analyzed some state-of-the-art SCs that implement them, aiming to cover a wide range of typical choices

XFRM. It is an integral part of the Linux kernel which provides a mechanism for implementing IPsec, enabling the enforcement of SPs and the establishment of SAs to protect IP traffic. XFRM supports various encryption, authentication, and key exchange algorithms, prioritising performance by operating directly at the kernel level[1]. Support for XFRM was included to cover scenarios relying on IPsec where strict standards must be met (e.g., site-to-site VPNs).

StrongSwan. It is an open-source implementation of the IPsec protocol suite. It supports IPv4 and IPv6 traffic, integrates the IKE protocol for negotiating SAs and key exchange, and has many plugins to extend its functionalities[2]. StrongSwan was supported to cover scenarios where the IKE protocol is handled directly and larger Virtual Private Networks (VPNs) deployments are required, with less direct user control in favour of automation. Like XFRM, StrongSwan is widely adopted for IPsec-based solutions, and many real-world cases involve their blended usage to achieve complete protection at the network layer.

OpenVPN. It is an open-source VPN solution designed to secure communication across various network environments using TLS. It supports multiple cryptographic algorithms and transport protocols, and employs the OpenSSL library to implement essential encryption, authentication, and key exchange mechanisms[3]. Support for OpenVPN was integrated to address secure channels established over the transport layer using TLS, acknowledging that it provides a more user-friendly approach at the expense of some performance.

C. Application scenario

As anticipated, one of our primary motivations is to provide a framework applicable in scenarios where the issues described in Section I are significant and the benefits we aim to bring would be valuable. We considered the context of *cloud security*, in which the discussed issues critically threaten information systems.

Cloud Security. Cloud-native computing represents a shift in application development, emphasising scalability and agility through containerised workloads, microservices architectures, and orchestration platforms like Kubernetes (K8s). Applications are decomposed into smaller services communicating over networks, enabling rapid development, deployment, and scaling. However, this microservices architecture introduces security challenges, including secure service-to-service communication, authentication, data protection, and dynamic workload management. Security in cloud-native environments requires ad-hoc solutions to address these challenges and ensure system protection.

III. The framework

The main contribution of this research is the CM-CP, a formal model of SCs for channel protection. It was developed using the Unified Modeling Language (UML) notation in Modelio[4], an open-source software able to export the diagram into XML Metadata Interchange (XMI), a machine-processable format. The model represents the fundamentals for executing the SCM for translating channel protection policies. This section presents how the SCM works and the base model on which the CM-CP was built.

A. General concept

In security-critical scenarios, security administrators must respond quickly and effectively to security events. They must counter cyberattacks before any damage occurs and identify vulnerabilities before attackers can exploit them. However, managing various technologies can be cumbersome due to their heterogeneity. This challenge led us to adopt a MDE approach, which emphasizes creating and using abstract models to automate the development and maintenance of complex systems. Our goal was to achieve maximum flexibility across different domains and to support a range of SCs by working solely at the model level.

In our vision, administrators should only define the security properties they want to enforce on communication channels from a high-level perspective. A translation tool should then automatically convert these definitions into configurations for a target SC, referred to as a Network Security Function (NSF) in virtual environments. Consequently, our SCM takes a secure communication policy as input and translates it into a configuration specific to the NSF. While ad-hoc GUIs can be more user-friendly, we adopted a validation PoC that receives a policy written in a vendor-independent Extensible Markup Language (XML) format. Despite its verbosity, we chose XML over JSON or YAML since XMI is XML-based.

All NSF-specific details are resolved automatically using grammar and semantics derived from the model. Notably, our design intentionally excludes any use of generative AI. While a formal model may seem outdated in the current landscape of Large Language Models (LLMs), we cannot accept the risk of errors or inaccuracies due to hallucinations. Nevertheless, in future works, the model could play a vital role in enhancing the reasoning of LLMs and minimizing mistakes.

[1] https://docs.cilium.io/en/latest/reference-guides/xfrm/index.html
[2] https://docs.strongswan.org/docs/latest/index.html
[3] https://openvpn.net/community-resources/how-to/
[4] https://www.modelio.org/

B. Base model

The base model was developed following Software Engineering (SE) best practices and established design patterns. This model, which has been presented for packet filters in previous works [4], [5], consists of two main components:

- *Capability Information Model*: defines the fundamental concepts related to security rules (e.g., conditions and actions), as well as policies (e.g., resolution strategies and default actions); it outlines the technical structure of security policies and tools without focusing on details;
- *Capability Data Model*: elaborates on the capabilities of different SCs; it includes hundreds of classes representing the operations of SCs of the same type, such as packet filters [5] and, as discussed in this work, secure channels.

Figure 1 illustrates the Information Model. The primary class is the *NSF*, which generically describes an NSF. It utilizes the decorator pattern to aggregate a collection of *SecurityCapability* instances, representing the security functionalities offered by the NSF. NSFs are gathered within the *NSFCatalogue*.

Additional classes are responsible for model-driven translation features. The *CapabilityTranslationDetails* class defines how security capabilities are translated into low-level language commands for specific NSFs. The *NSFPolicyDetails* class provides further information on creating specific rules and configurations for a particular NSF. Lastly, the *CapabilityGroup* class specifies groups of capabilities that are functionally linked within each NSF (more details in Section V-B).

The *SecurityCapability* class has been subclassed to define essential concepts for modeling policies, based on an analysis of various SCs. Three key concepts are necessary for effective rule modeling, as shown in the same Figure. The *conditions* refer to features of the NSF that determine the targets of rule enforcement, which include constraints like IP addresses or regex matching on HTTP types. The *actions* denote operations that the NSF can perform, such as denying packets or encrypting flows. The *evaluations* are formulas that dictate rule activation based on specific conditions.

Additionally, two concepts are critical for constructing security policies. The *resolution strategies* describe how the SCs reacts when multiple rules are triggered, such as prioritizing actions from the highest priority rule. The *default actions* indicate the behavior of the SCs when no rules are activated, such as implementing a `deny all` strategy.

C. Framework Components

Figure 2 illustrates the general structure and workflow of the SCM. It utilizes the following artefacts:

- *XMI Model*: the model in XMI format, which includes all the security capabilities for every supported NSF;
- *NSF Catalogue*: contains explicit information about the supported NSFs, along with the classes that define how to translate the security capabilities for each NSF;
- *XSD Capability Data Model*: includes the grammar (defined as XML Schema Definition (XSD)) necessary for validating all entities in the *NSF Catalogue* against the structure defined in the model;
- *XSD Grammar for an NSF*: defines the grammar to validate an abstract policy for a target NSF; it consists of a subset of the XSD definitions from the *XSD Capability Data Model* for an NSF;
- *XML Policy*: contains the rules written in the abstract notation that will be translated for a target NSF;
- *Low-Level Policy*: contains the security configuration defined in the *XML Rule*, translated for a target NSFs.

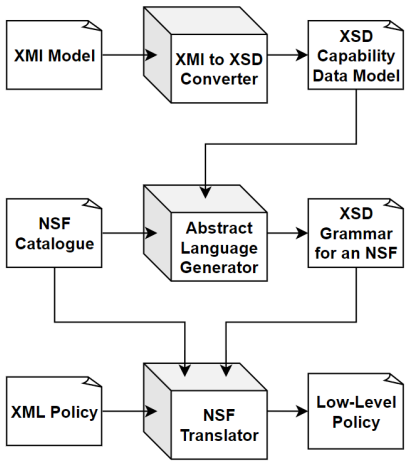

Fig. 2. Translation Flow

When a new UML model version becomes available, typically to support features of new categories of NSFs, the *XMI to XSD Converter* is used to process the updated *XMI Model*. This tool generates the *XSD Capability Data Model* as its output, and it is executed only occasionally.

Once the *XSD Capability Data Model* is generated, or whenever new information about an NSF is added to the *NSF Catalogue*, the *Abstract Language Generator* must be executed. This tool takes the *XSD Capability Data Model*, the *NSF Catalogue*, and the name of an NSF as inputs. Its output is the *XSD Grammar for an NSF* for that specific NSF. Like the previous tool, the *Abstract Language Generator* is not frequently executed.

Finally, when an *XML Policy* is ready to be enforced for a specified NSF, the *NSF Translator* generates the *Low-Level Policy*. For this purpose, it requires the *XSD Grammar for an NSF* and the translation instructions in the *NSF Catalogue*. This tool is the most frequently executed, since it is needed whenever high-level security requirements change or when translation details for the involved capabilities are updated. For instance, this tool will be utilized if a notation for a supported functionality related to a NSF is officially deprecated and replaced with a new one.

IV. The Capability Model - Channel Protection CM-CP

This section will present in depth the CM-CP. It will also analyze the methodology adopted to reach a stable version.

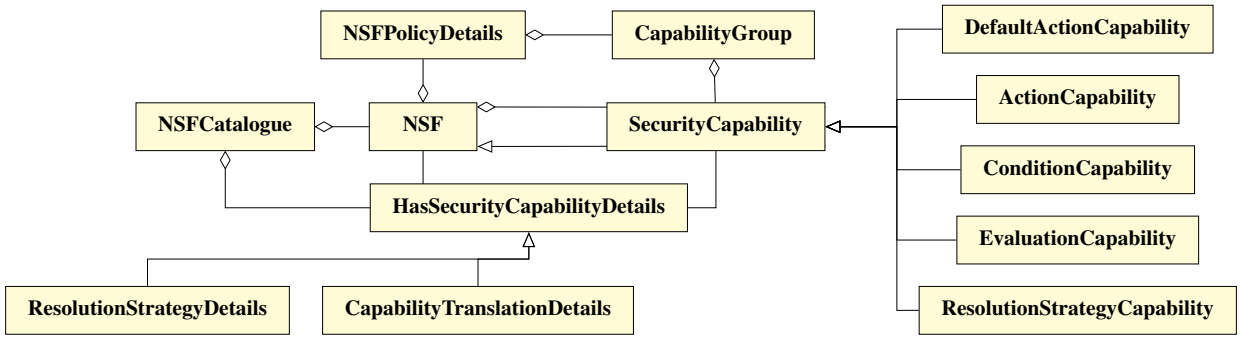

Fig. 1. The Security Capability Information Model (CapIM)

A. General requirements

Since the model of channel protection constitutes the fundamentals of the work, it was mandatory to approach the design of the CM-CP following some general but essential goals:

- *abstraction layer*: the model must introduce an intermediate layer at which channel protection NSFs are depicted in a generalised form, enabling a high-level description of security properties regardless of the enforcing technology;
- *extensibility*: the model must be flexible, allowing for adding or modifying existing mechanisms and techniques without unjustified effort;
- *modularity*: the model must maintain a modular conception to guarantee that operating on a sub-component does not influence external components;
- *backward compatibility*: the model must remain aligned with prior works, respecting established architectural decisions whenever possible; performing changes must be justified and limited to the necessary scope;
- *support for real-world use cases*: the model must address practical concerns emerging from real-world scenarios; it must be possible to map real configurations without neglecting any security property.

B. Methodology

A foundational assumption is that a thorough study of the target domain is essential to understand the nature of the security operations implemented in the model. We observed that updating the model and the related framework artifacts through a single, waterfall-like process is impractical. Therefore, we adopted an iterative process to add support for the new domain. Below are outlined the main steps of this approach:

1) *identify a stable subset of relevant security properties*: perform a precise analysis of functionalities in representative NSFs (e.g., StrongSwan for channel protection), emphasizing core concepts over implementation details;
2) *model and add the identified properties*: rigorously model and integrate the identified requirements into the main model using appropriate tools (e.g., Modelio); validation against reference and external NSFs is crucial; any new technical functionality necessary for the translation process should be documented for future reference;
3) *update framework artifacts related to translation management*: export the new version of the model as XMI (i.e., *XMI Model* artifact) and expand the *NSF Catalogue* with new security capabilities and low-level details;
4) *modify the translation procedure as needed*: if *step 2* indicated the necessity for adjustments to the translation process, these changes must be implemented; this involves updating the *NSF Translator* to accommodate new operational requirements, such as managing dependencies among capabilities or groups of capabilities (discussed in Section V);
5) *conduct extensive testing of all additions*: perform thorough testing, including varying-size translation trials, in which the framework processes abstract configurations and the outputs are validated against state-of-the-art solutions (i.e., the NSFs for which support has been added);
6) *repeat from step 1*: this iterative process should continue until the desired domain coverage is fully achieved.

These steps have been presented in a generic format to evaluate their potential reuse in various expansion efforts beyond channel protection.

C. Channel Protection Data Model

The CM-CP introduces the fundamental elements required to secure a communication channel. Figure 3 captures a small subset of the model's classes, consisting of the main components of protected communication, namely:

- *EncapsulationTechniqueCapability* defines how data in transit is enveloped before transmission;
- *EncryptionTechniqueCapability* defines how data in transit is protected to ensure confidentiality;
- *DataAuthenticationTechniqueCapability* defines how data in transit is protected to ensure data authentication;
- *PeerAuthenticationTechniqueCapability* defines how the communicating parties authenticate each other;
- *AETechniqueCapability* defines how data in transit is protected using authenticated encryption algorithms (i.e., algorithms that concurrently provide confidentiality and data authentication);
- *KeyMaterialCapability* defines the algorithms and techniques the parties use to exchange secrets and parameters for channel protection. This can be achieved statically or dynamically. For a dynamic exchange, the *DynamicKeyExchangeCapability* class is directly associated with the *ChannelProtection* class, as the exchange occurs over the

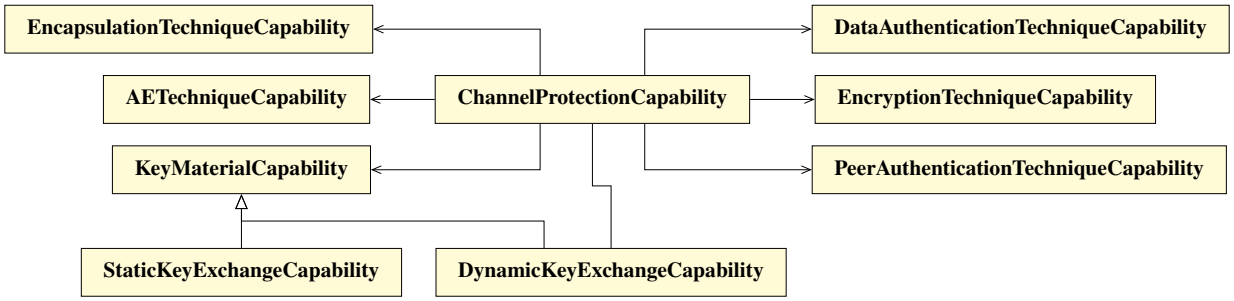

Fig. 3. Capability Model - Channel Protection (CMCP)

same channel. In contrast, the *StaticKeyExchangeCapability* class is not directly related to the *ChannelProtection* class, as it usually takes place Out-Of-Band (OOB).

Overall, the CM-CP contains 35 other classes. They were all defined as subclasses of the ones described above. A picture of the complete diagram is available at our repository[5].

V. Modelling Channel Protection: Open Issues

This section presents the issues we faced regarding the modeling of peculiar techniques in the channel protection domain, the solutions adopted, and why they proved reasonable. Furthermore, we addressed the final performance of the framework by measuring the duration of translations for different target NSFs and dimensions of the policies.

The latest analyzed NSFs (e.g., strongSwan) introduced peculiar formats of security operations. For instance, configuration parameters can be gathered into groups to express more advanced features that need to be enforced. Furthermore, the effects of specific requirements might be influenced by others, meaning that some operations are connected and depend on each other. These emerging aspects suggest that requirements in real security configurations can be more complicated than simply listing the operations in sequence without considering the surroundings. Therefore, a stateful description of the language peculiarities using *dependencies* among capabilities and *groups* of different capabilities is necessary.

A. Dependencies among capabilities

Channel protection SCs provide more sophisticated security features than other domains (e.g., packet filters). Typically, the one-rule-per-line format is unsuitable because secure communication rules can span across various statements, whose effects influence each other. For instance, when specifying data protection for a secure channel, the definition of the encryption property is allowed only if the protocol type is the Encapsulating Security Payload (ESP). Again, these interactions among properties must be integrated at the model level. The solution available in literature did not provide adequate support for dependencies [5], so a new technical class was defined: *CapabilityDependency*. This new class can be instantiated multiple times for each security capability, and it is used to describe how a capability depends on others in the scope of one NSF. The class exposes the following attributes for dependency management:

- *capabilityName*: the name of the capability on which the current one depends;
- *capabilityValue*: the optional value of the capability in *capabilityName*;
- *dependencyType*: the value of the dependency type. The model supports *PRESENCE_OF_CAPABILITY* and *ABSENCE_OF_CAPABILITY* to state that the capability in *capabilityName* must be present or absent in the abstract rule to use the current one; the *PRESENCE_OF_VALUE* and *ABSENCE_OF_VALUE* allow for the use of the current capability only when the one in *capabilityName* does or does not have the specific *capabilityValue*

Listing 1 shows how dependencies are specified. Unlike other issues, the integration of dependency management necessitated important updates to the SCM. This adjustment was essential, as evaluating dependencies among capabilities is feasible only while translating a *Low-Level Policy*. To address this need, a specific structure, the Partially Ordered Set (POSET), was introduced within the SCM. This structure gathers potential combinations of dependencies among capabilities. All POSETs associated with a particular NSF are generated and organized into a *dependency tree*. During the translation of a *Low-Level Policy*, this dependency tree is traversed to locate a POSET embodying the dependencies of the rule. The translation progresses if such a POSET is identified. Conversely, if it is not found, the *NSF Translator* raises an exception, indicating that specific dependencies have not been satisfied. This approach was proven efficient, as the generation of the dependency tree for an NSF is resource-demanding, but occurs only when producing the abstract language of the NSF, which is infrequent. In contrast, the subsequent translation process occurs more regularly and is significantly expedited.

B. Groups of capabilities

The introduction of *groups* of capabilities can be illustrated by the example of StrongSwan. In this context, certain security options are pertinent only when applied within specific scopes. For instance, when establishing a VPN between two subnets, StrongSwan enables users to specify details about both the local and remote subnets from the perspective of the device managing one of the endpoints. Rather than providing pairs of similar options differing solely for the endpoint they reference

[5]https://github.com/DaveC0la/SCMPolicies.git

```
<capabilityTranslationDetails>
 <nSF ref="IpTables"/>
 <securityCapability ref="
     AhReservedConditionCapability"/>
 ...
<capabilityDependencies>
 <capabilityDependency>
  <capabilityName>IpProtocolTypeConditionCapability
      </capabilityName>
  <capabilityValue>AH</capabilityValue>
  <dependencyType>PRESENCE_OF_VALUE</dependencyType>
 </capabilityDependency>
</capabilityDependencies>
 ...
</capabilityTranslationDetails>
```

Listing 1. Capability dependencies

(local or remote), StrongSwan utilizes the concept of groups within its configuration files. This method is adopted in many advanced security tools and is not limited to secure channels. It facilitates the definition of a single security option for each functionality, which can subsequently be assigned to various operational contexts based on the group to which it belongs.

Following the MDE approach, a group of capabilities can be declared for an NSF with a *capabilityGroup* class instance, as shown in Listing 2. It consists of:

- *capabilityGroupName*: the name of the group of capabilities for the current NSF;
- *capabilityGroupStart*: the item to place at the beginning of the group of capabilities in the low-level policy;
- *capabilityGroupEnd*: the item to place at the end of the group of capabilities in the low-level policy;
- *capabilityGroupConcatenator*: the item to place between the capabilities of the group in the low-level policy

```
<nSF id="StrongSwan">
 <nsfPolicyDetails>
  <ruleStart>ike-name {\n\t</ruleStart>
  <ruleEnd>\n}\n</ruleEnd>
   ...
  <capabilityGroup>
   <capabilityGroupName>local</capabilityGroupName>
   <capabilityGroupStart>local {\n\t\t</
       capabilityGroupStart>
   <capabilityGroupEnd>}\n\t</capabilityGroupEnd>
   <capabilityGroupConcatenator>\t</
       capabilityGroupConcatenator>
   <securityCapability ref="
       SourceIdentityConditionCapability"/>
   ...
  </capabilityGroup>
 </nsfPolicyDetails>
 ...
 <securityCapability ref="
     SourceIdentityConditionCapability"/>
 ...
</nSF>
```

Listing 2. Capability groups

The elements that belong to a group are specified as *securityCapability* elements, and must also belong to the list of capabilities supported by the NSF (i.e., it is not possible to assign a security capability to a group for one NSF if that NSF does not support that capability in the first place). The model poses no limit to the number of *capabilityGroup* elements in the *nsfPolicyDetails* (i.e., there is no maximum amount of capability groups that can be defined for an NSF).

The POSET emerged as the most suitable solution to handle groups of capabilities during translation, as it already dealt with the dependencies among capabilities. When the abstract language for an NSF is generated (through the *XSD Language Generator* tool), the dependency tree is also generated (as stated in section V-A); information about groups is extracted and processed into additional constraints to further refine the dependency tree and the related POSETs.

C. Analysis of the performances

This section proposes an overview of the performances for the SCM's tools discussed in Section III-C. The goal is to observe the execution time of the main processes after the new contributions to the model from a practical perspective. The tools were executed with an automated script and averaged over 30 runs to mitigate the impact of system load or other external factors. All measurements were performed on a PC with an *AMD Ryzen 7 5825U* processor and 16 GB of RAM.

XMI to XSD Converter. The execution time for the model conversion from XMI to XSD was, on average, *18.2* seconds. This aligns with previous versions of the SCM, as expected. The *XMI to XSD Converter* processes the XMI model, which is now expanded since it includes the CM-CP. Still, the efficiency of the tool is sufficiently high to handle the model's expansion without any significant impact on performance.

Abstract Language Generator. The next measurement focused on the generation of the XSD specific to each NSF introduced in this work: XFRM, StrongSwan, and OpenVPN. The execution times of the *Abstract Language Generator* for these NSFs were *2.5*, *2.8*, and *2.4* seconds, respectively. No significant variation was observed among them. However, these times are slightly higher than those recorded in earlier versions of the SCM, which is attributable to the inclusion of the POSET structure. This structure is computed during this step and retained for the subsequent policy translation phase. Nonetheless, the efficiency of the tool remains high. In fact, despite StrongSwan requiring more complex structural handling, its XSD generation time is nearly identical to that of XFRM and OpenVPN, indicating that the additional complexity does not meaningfully degrade execution performance.

NSF Translator. The final measurements concerned the translation time of different security policies across the channel protection NSFs that are now supported by the model (i.e., XFRM, StrongSwan, and OpenVPN). This aimed to assess how efficiently the CM-CP enables the SCM to translate security policies for our domain of interest. To do this, a representative security rule was defined and repeated in the policy for a fixed number of occurrences (i.e., 1, 10, and 100). For consistency, the core structure of the rule remained similar across all NSFs, with adjustments only where a given capability was unsupported by one NSF. Each base rule included approximately 15 capabilities to provide a uniform baseline for comparing translation times. The *NSF Translator* was executed and the results are presented in Table I.

It can be observed that for each NSF, when the number of rules in a policy increases by a factor of *10*, the translation time

TABLE I
AVERAGE TRANSLATION TIME (IN SECONDS) OVER 30 RUNS

Technology	1 Rule	10 Rules	100 Rules
XFRM	1.7	4.7	37.0
StrongSwan	2.6	17.1	180.7
OpenVPN	1.5	5.2	45.3

does not scale by the same factor. Indeed, the time required to translate a policy with a single rule is disproportionately high compared to policies with 10 or 100 similar rules, which show a sublinear increase in processing time. This behavior can be justified by reasoning about how the translation process works with the new approach. As discussed previously, the POSET is a complex data structure introduced to address the open issues in policy translation for secure channels, and is used to translate the policies accordingly. The key insight is that it must be loaded into memory prior to the main processing, and this initialization introduces a fixed computational overhead. However, since the POSET is loaded only once per policy and reused across all rules in the policy, this overhead becomes increasingly negligible as the number of rules grows.

Another interesting observation is the significantly higher translation time for StrongSwan policies compared to those for XFRM and OpenVPN. This can be attributed to differences in the complexity of their configuration languages. While XFRM and OpenVPN follow relatively simple, sequential structures for defining security configurations, StrongSwan requires more sophisticated structures and mechanisms, such as grouping of capabilities and advanced dependency management. As a result, the *NSF Translator* must process more metadata and perform additional operations in the POSET to generate a valid low-level policy for StrongSwan.

VI. VALIDATION

The outcomes of this research underwent an evaluation to prove the coverage of the CM-CP in capturing features exposed by channel protection NSFs. The goal was to confirm that the proposed solution simplifies the management of these NSFs. From our perspective, the realization of the model is deemed successful if it achieves an adequate level of expressiveness regarding security capabilities for communication channels. Additionally, we selected a concrete scenario as a benchmark to verify the enforceability of the configurations.

A. Validation of the expressiveness

Assessing the CM-CP model's expressiveness meant to evaluate how effectively it defines security requirements across the supported channel protection NSFs. To verify the coverage of the model, we first investigated whether all operations in the XFRM, StrongSwan, and OpenVPN configuration languages for securing a communication channel have a capability counterpart in the CM-CP. Then, we realized a series of abstract policies to validate the individual capabilities.

Specifically, we verified the ability of the model to represent real configurations by manually converting most of the StrongSwan examples from the official documentation[6] into abstract policies. Each example includes StrongSwan configuration files for specific scenarios and the resulting XFRM rules in the kernel. Then, these abstract rules were translated back into strongSwan and XFRM rules by the SCM, certifying the correctness with the initial ones. This way, the original security properties of the examples provided by this guide were verified to comply.

We also evaluated the extensibility of the CM-CP. Initially designed taking XFRM and StrongSwan as validating references, support for OpenVPN was introduced later with minimal updates to the SCM (i.e., translation details for OpenVPN rules and related security capabilities) and no changes to the model. After the extension, we repeated the expressiveness validation for OpenVPN. The reference manual[7] was taken to forge configuration policies for the *openvpn* daemon, considering similar scenarios to those addressed while validating XFRM and StrongSwan. Then, we mapped these policies onto model-compliant abstract policies using the abstract notation of the framework and translated them back to configuration files for OpenVPN, assessing total reversibility of the process.

As a last step, we verified the model's ability to address multiple NSFs with a single abstract policy. This is possible only when the target NSFs share the capabilities in the abstract policy[8]. The SCMPolicies[9] repository can be referred to for a comprehensive set of translation examples.

B. Validation in real-world scenarios

The second validation phase involved testing its applicability in a concrete infrastructure. The goal was to verify the correctness of the configurations generated through the SCM through the model. We considered representative aspects from the cloud domain introduced in Section II. For each, we discussed typical operations for securing communication channels and conceived the required security capabilities at a high level, defining them with the abstract notation.

These requirements should generally be directly refined into low-level configurations for a certain NSF. Conversely, our proposed approach allows refinement into an abstract policy, which is then automatically translated into ready-to-deploy configurations for one of the available NSF. The deployment was conducted in a virtual environment for validation purposes, as Figure 4 exemplifies.

To emulate a two-site VPN, Linux network namespaces were used to inspect every packet on the wire. Four namespaces reproduce the topology of a small enterprise: *Host-A* (10.0.1.2) and *Host-B* (10.0.2.2) act as ordinary hosts on *LAN-A* (10.0.1.0/24) and *LAN-B* (10.0.2.0/24), respectively, while *GW-A* and *GW-B* serve as their security gateways, each carrying a LAN-side address (10.0.1.1 and 10.0.2.1) and a

[6] https://docs.strongswan.org/docs/latest/config/IKEv2.html
[7] https://openvpn.net/community-resources/reference-manual-for-openvpn-2-6/
[8] If an operation in a policy is implemented at a low level for one NSFs but not for another one, it is conceptually inaccurate to request translation for both of them
[9] https://github.com/DaveC0la/SCMPolicies.git

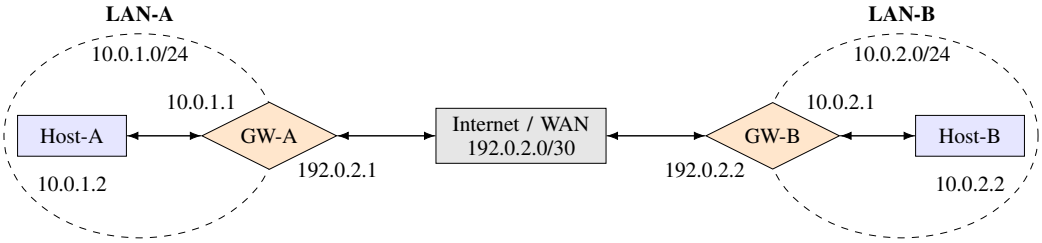

Fig. 4. High-level network topology

WAN-side address (192.0.2.1 and 192.0.2.2). The physical layout is recreated in RAM using veth pairs, giving a controllable infrastructure in which strong-security tunnels can be validated against cloud-like network conditions. Furthermore, plausible network impairments (e.g., non-negligible round-trip time, delay, etc.) were applied to the gateway-to-gateway simulated WAN connection.

To approximate a real-world public-cloud environment, the abstract policies delivered to the SCM expose capabilities that map directly to cloud providers' VPN features. Recommended settings for *encryption* and *authentication* algorithms, *dead-peer detection (DPD)*, and packet handling, such as *fragmentation*, follow the recommendations provided by well-established vendors, specifically Amazon Web Services (AWS)[10] and Microsoft Azure[11], thereby aligning the experimental setup with current industry best practice.

The SCM was then instructed to translate the abstract policies into StrongSwan-specific configurations (assuming that a network administrator chooses the proper implementation for a VPN). These were enforced in the gateways, and adequate testing on the wire was performed to assess the correctness of the properties. A guide to replicate the validation infrastructure and observe the measurements is situated at our repository[12].

The experiment confirms that the CM-CP captures a wide range of security requirements. Network administrators should only select the properties they wish to enforce in the high-level domain. Once the target NSF is selected, the abstract policy is processed automatically and converted into the proper low-level configuration. Currently, no tool supports writing abstract configurations, e.g., with a user-friendly GUI. Consequently, future work will investigate advanced tools to generate abstract policies automatically based on natural language intents.

VII. Related works

To reflect upon our research, we analyzed concurrent solutions and research works conducted in similar scopes and with related approaches. Exploiting security intent-based translation has recently proven to be a successful research path, with many authors and researchers carrying out novel solutions.

Leivadeas et al. [6] recently proposed a literature review concerning Intent-Based Networking (IBN). They discuss how IBN simplifies network configuration and management, noting the lack of a unified vision and set of conventions in this direction. Their review points out many approaches that can be advantageous in intent profiling and translation, citing the usage of intent-based languages and model-driven translation (i.e., techniques we adopted in our research).

Bringhenti et al. [7] propose a solution for the automated configuration of security-critical tools (namely, firewalls and secure channels) using a high-level representation of the network to be secured. This is one of the inputs provided to a solver (i.e., Z3) to configure security policies automatically. The usefulness of this approach is justified concretely by the authors, who also acknowledge the necessity of providing real network scenario validation. Similar studies have been conducted by Landeau et al. [8]. They propose an introduction to a novel algorithm for automated intent placement (i.e., PSNIP), which uses the abstract representation of the infrastructure. These proposals address the necessity of reducing human intervention and dependence on low-level conventions. Still, they concentrate more on optimising the policies, starting with the provided network topologies. Conversely, we focus on representing security policies, but concerning the SCs exposed by security tools offered by the many vendors.

Other researchers have explored ways to define high-level intents, but with approaches that, while valuable and insightful, diverge from our goals. Xiao et al. [9] present a mechanism to translate user-defined intents using a pure Natural Language Processing (NLP) approach. They cite the drawbacks of having custom configuration languages to motivate their work while acknowledging that their translation success maintains margins of improvement. Our work shares motivations (i.e., the automatic transition from high-level to low-level requirements). Still, we choose the path of intent modelling coupled with a custom abstract notation because we want to prioritise the correctness of the translation. This choice is due to the security-critical nature of the systems we target. Similarly, Van Tu et al. [10] analyse a solution to translate intents defined by the user into a JSON file using LLMs (justified with their efficiency in managing JSON schema). Again, the motivations are similar, but, departing from our target (i.e., an enforceable configuration with no need for further human intervention), their translation stops at a higher-level step.

The benefits of the work we propose can be applied to native cloud scenarios, as documented by Theodoropoulos et al. [11]. Their study highlights a set of cyberattacks targeting communication channels in cloud environments, such as Man-

[10] https://docs.aws.amazon.com/vpn/latest/s2svpn/VPC_VPN.html
[11] https://learn.microsoft.com/en-us/azure/vpn-gateway/
[12] https://github.com/DaveC0la/SCMPolicies.git

In-The-Middle (MITM) attacks. Moreover, it emphasises the importance of process automation, which reduces human errors, ensures consistency in configurations, and enables rapid responses to threat events. The critical role of communication in cloud scenarios is also underscored by Saxena et al. [12], who propose a model to monitor connections among virtual machines that collaborate in providing cloud services. Their approach focuses on identifying and eventually terminating suspicious inter-machine links. Similarly, Fabian Süß et al. [13] emphasise the significance of addressing threats to communication solutions in cloud environments. Their score-based analysis of cyberattacks reveals medium-to-high-risk scores for metrics related to secure channels and data transmission, confirming their importance in cloud infrastructures.

VIII. Conclusions

This paper presented our research on modeling channel protection security and the automatic translation for diverse technologies. The objective was to eliminate reliance on specific solutions, thus achieving a vendor-agnostic approach and reducing human errors.

The primary contribution is the CM-CP, a formal model that abstracts the secure communications features provided by SCs. This model enables high-level descriptions of the techniques used to protect communication channels. A translation framework is also proposed, named SCM. Its purpose is to leverage the CM-CP model for processing high-level security policies and converting them into low-level configurations suitable for target implementations. This process occurs automatically, without requiring user intervention. The key outcome is that network administrators can manage channel protection from a high-level perspective, allowing them to neglect specific technical details. As a result, they can concentrate on other responsibilities while significantly reducing the likelihood of errors due to complex configurations. The expressiveness of the CM-CP was validated against state-of-the-art communication solutions, namely XFRM, StrongSwan, and OpenVPN. Furthermore, several instances of abstract security policies were successfully translated, and a concrete infrastructure that emulates a real-world scenario was instantiated as an enforcing benchmark for those translations, demonstrating the correctness and rigor of our approach.

Future work will enhance support for other SCs and introduce more capabilities, thereby broadening the model's coverage for channel security. Moreover, a GUI plugin could improve accessibility to the high-level XML notation. Although XML is convenient for the reasons outlined in the paper, it could benefit from a way to overshadow its verbosity. Simultaneously, developing an abstract policy generator utilizing LLMs is under consideration. This system should interpret natural language requests related to desired security features and draft a high-level security policy that fulfills these requirements, which would serve as input for the SCM. The model will function as a training and validation tool to reduce hallucinations. This direction, aiming for fully automated intent-based security management, will address the learning curve and verbosity of the XML abstract notation, as an autonomous system will entirely manage the high-level policy.

IX. Acknowledgments

The work of Davide Colaiacomo and Cataldo Basile was carried out within the SERICS – Security and Rights in the CyberSpace project and received funding from the European Union Next-GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) - MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.3 - D.D. 1556 11/10/2022, PE00000014). It is also supported by the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme under Grant Agreement No. 101139198, iTrust6G project. Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or SNS-JU. Neither the European Union nor the European Commission nor the granting authority nor the Ministry can be held responsible for them.

References

[1] S. Frankel and S. Krishnan, "IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap," RFC 6071, Feb. 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6071

[2] C. Kaufman, P. E. Hoffman, Y. Nir, P. Eronen, and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)," RFC 7296, Oct. 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7296

[3] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8446

[4] C. Basile, G. Gatti, and F. Settanni, "A formal model of security controls' capabilities and its applications to policy refinement and incident management," 2024. [Online]. Available: https://arxiv.org/abs/2405.03544

[5] C. Basile, D. Canavese, L. Regano, I. Pedone, and A. Lioy, "A model of capabilities of network security functions," in *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*, 2022, pp. 474–479.

[6] A. Leivadeas and M. Falkner, "A survey on intent-based networking," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 625–655, 2023.

[7] D. Bringhenti, R. Sisto, and F. Valenza, "Automating the configuration of firewalls and channel protection systems in virtual networks," in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, 2023, pp. 474–479.

[8] G. Landeau, M. Avgeris, A. Leivadeas, and I. Lambadaris, "Security-oriented network intent placement using particle swarm optimization," in *2023 7th Cyber Security in Networking Conference (CSNet)*, 2023, pp. 19–22.

[9] Y. Xiao, W. Quan, H. Zhou, M. Liu, and K. Liu, "Lightweight natural language driven intent translation mechanism for intent based networking," in *2022 7th International Conference on Computer and Communication Systems (ICCCS)*, 2022, pp. 46–51.

[10] N. Van Tu, J.-H. Yoo, and J. W.-K. Hong, "Towards intent-based configuration for network function virtualization using in-context learning in large language models," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, 2024, pp. 1–8.

[11] T. Theodoropoulos, L. Rosa, C. Benzaid, P. Gray, E. Marin, A. Makris, L. Cordeiro, F. Diego, P. Sorokin, M. D. Girolamo, P. Barone, T. Taleb, and K. Tserpes, "Security in cloud-native services: A survey," *Journal of Cybersecurity and Privacy*, vol. 3, no. 4, pp. 758–793, 2023. [Online]. Available: https://www.mdpi.com/2624-800X/3/4/34

[12] D. Saxena and A. K. Singh, "Osc-mc: Online secure communication model for cloud environment," *IEEE Communications Letters*, vol. 25, no. 9, pp. 2844–2848, 2021.

[13] F. Süß, M. Freimuth, A. Aßmuth, G. R. S. Weir, and B. Duncan, "Cloud security and security challenges revisited," 2024. [Online]. Available: https://arxiv.org/abs/2405.11350