CCSM: Building Cross-Cluster Security Models for Edge-Core Environments Involving Multiple Kubernetes Clusters

ABSTRACT

With the emergence of 5G networks and their large scale applications such as IoT and autonomous vehicles, telecom operators are increasingly offloading the computation closer to customers (i.e., on the edge). Such edge-core environments usually involve multiple Kubernetes clusters potentially owned by different providers. Confidentiality concerns could prevent those providers from sharing data freely with each other, which makes it challenging to perform common security tasks such as security verification and attack/anomaly detection across different clusters. In this work, we propose a solution for building cross-cluster security models to enable various security analyses, while preserving confidentiality for each cluster. We design a six-step methodology to model both the cross-cluster communication and cross-cluster event dependency, and we apply those models to different security use cases. We implement our solution based on a 5G edge-core environment that involves multiple Kubernetes clusters, and our experimental results demonstrate its efficiency (e.g., less than 8 seconds of processing time for a model with 3,600 edges and nodes) and accuracy (e.g., more than 96% for cross-cluster event prediction).

1 INTRODUCTION

Cloud and Kubernetes clusters have become standard and commonplace solutions enabling more cost-effective deployment of 5G applications. At the same time, to support large-scale applications such as IoT and autonomous vehicles, telecom operators are increasingly offloading the computation closer to customers (i.e., on the edge) in order to satisfy the latency and throughput requirements [15, 41]. While these strategies improve the overall performance and quality of service, security is often an afterthought: the distribution of workload among multiple clusters may increase the attack surface, the offloading or extending of the cloud may potentially involve less trusted or less protected edge providers. Moreover, confidentiality concerns of the providers may prevent them from sharing data freely with each other [38], particularly in scenarios where data sovereignty and cross-border data transfers are involved or when no prior trust relationship has been established.

There exist various security solutions for clouds and Kubernetes clusters, such as security verification [45], security impact prediction [63], and attack/breach detection [2] (a more detailed review of related work is given in Section 10). However, most such solutions are designed for enforcing security locally at each cluster, and they cannot be easily extended across multiple (edge and core) clusters. Furthermore, as mentioned above, the providers of those clusters would be reluctant to disclose confidential or private information about their infrastructure and users. This makes it infeasible to apply those existing security solutions on a central copy of data from all the clusters, which is necessary for many security analyses, such as verifying cross-cluster security breaches, detecting cross-cluster attacks, or predicting events across multiple clusters. To make things worse, as containers and cloud-native computing [13] are widely adopted due to their clear advantages in terms of less overhead and better performance, these also suffer from buggy images and weaker isolation compared to full-fledged VMs. For this reason, container environments and container orchestrators (such as Kubernetes) have become attractive targets of various security attacks [62]. Considering that more and more providers of critical services, including 5G core Network Functions (NFs) [21], are moving to the cloud, addressing those limitations is a pressing concern.

Motivating Example. To make our discussions more concrete, we present a motivating example in the context of 5G networks. Specifically, Fig. 1 depicts a typical 5G edge-core environment, where the core cluster is owned by a mobile network operator who provides private 5G services [54, 57] as a managed service to two different vertical industries, *Company 1* and *Company 2*, who own the two edge clusters. The two General Data Protection Regulation (GDPR) [50] icons attached to the two edge clusters indicate confidentiality concerns about leaking user information, which prevent *Company 1* and *Company 2* from sharing their data freely with the mobile network operator. On the other hand, they have to rely on the operator to provide necessary 5G core services and functionalities, including security solutions.

In particular, suppose the administrators of the two edge clusters would like to verify a given security policy that their services are completely isolated from each other. For this purpose, as demonstrated by the call-outs, the administrators build graphic models about communications inside each cluster. Looking at each such local model alone, each administrator is convinced that the policy is satisfied since all the communications are limited to be between network functions inside the cluster. Similarly, the core administrator also sees nothing wrong in his/her local model (note the two network functions shown in gray color are not a concern as it is quite normal for some resources in the core to be shared [31]).

However, as demonstrated in the call-out at the bottom, relying solely on local models is insufficient to capture potential cross-cluster security breaches. Specifically, both edge administrators fail to see the fact that one of the network functions in *Edge 1* can actually reach another network function in *Edge 2* through the core (e.g., due to a misconfiguration in the AUSF and SMF2 network functions). This violation of the given security policy cannot be identified based on the local models alone, and is only visible in a global model comprising information across all three clusters, as shown in the figure. Nonetheless, the aforementioned confidentiality concerns mean that such a global model cannot be trivially built by collecting data from all clusters. Instead, a solution must carefully balance between the need to build the global model to enable security solutions and the need to preserve confidentiality for each cluster.

Our solution. We propose an approach to build cross-cluster security models (CCSM) to address the aforementioned challenge. Specifically, we design a methodology to (i) construct local security models at each edge cluster, and extend them to establish external links to other clusters, (ii) prune and anonymize the local models

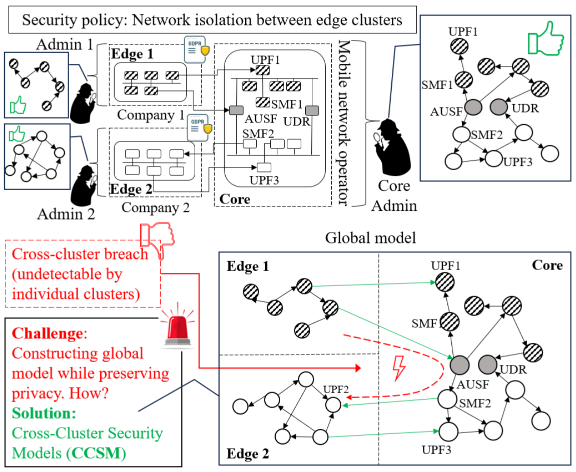

Security policy: Network isolation between edge clusters
Admin 1
Edge 1
Company 1
UPF1
SMF1
AUSF
UDR
SMF2
UPF3
Mobile network operator
Core
Admin
Edge 2
Admin 2
Company 2
Core
UPF1
SMF1
AUSF
UDR
SMF2
UPF3
Cross-cluster breach (undetectable by individual clusters)
Global model
Edge 1
UPF1
Core
SMF
UDR
AUSF
UPF2
SMF2
Edge 2
UPF3
Challenge: Constructing global model while preserving privacy. How?
Solution: Cross-Cluster Security Models (CCSM)

Figure 1: Motivating example

to avoid unnecessary information sharing before sending them to the core cluster, and (iii) construct and prune the global model at the core cluster before sending it back to each edge cluster. While this general methodology can potentially be applied to many different security models, we demonstrate such applicability through instantiating it to capture two complementary 5G security perspectives, namely, the communication among 5G network functions and the dependency among 5G events. Furthermore, we show how those security models may enable verifying cross-cluster security breaches, detecting cross-cluster attacks/anomalies, and predicting cross-cluster security impacts. As a proof of concept, we implement the solution and integrate it with Kubernetes [39] and Free5GC [27] (a popular open-source 5G network implementation), and evaluate its effectiveness and efficiency through experiments.

Contribution. In summary, our main contributions are as follows:

- This work enables a wide range of cross-cluster security solutions. Specifically, the proposed methodology can construct a global view of multi-cluster environments while preserving the confidentiality of each cluster. Such a global view contains important cross-cluster information (which are invisible at each cluster) that could enable various security solutions to identify cross-cluster security breaches, cross-cluster attacks/anomalies, and cross-cluster security impacts, which would otherwise be hidden if examined locally at each individual cluster.
- The proposed methodology consists of six steps for integrating clusters' local views in a confidential and scalable manner, and distributing the constructed global view back to each cluster on a "need to know" basis. This general methodology is instantiated to capture both the communication among 5G network functions and the dependency among 5G events, which complement each other to provide more security perspectives. To demonstrate the applicability of our solution, we describe three potential applications of our solution including security verification, attack/anomaly detection, and security impact prediction.
- To evaluate our solution, we generate the first multi-cluster dataset covering both communications among 5G core network functions and 5G control plane events in an edge-core testbed environment involving multiple inter-connected Kubernetes clusters. We integrate our solution into Free5GC, an open-source 5G core implementation, and Kubernetes clusters. We evaluate our solution through experiments which demonstrate its efficiency (e.g., less than 8 s to process models with 3,600 edges and nodes) and accuracy (e.g., more than 96% to predict cross-cluster events).

2 PRELIMINARIES

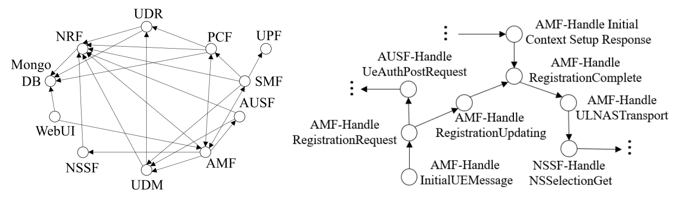
This section provides necessary background and our threat model.

Edge-Core Model and Private 5G. In the context of 5G networks, the edge-core model is a distributed architecture allowing a provider to move some resources and services closer to the user (i.e., at the *edge* of the network) while keeping core functions in a central place (i.e., at the *core*). An example of this model is shown in Appendix A (due to space constraint). This allows better throughput and latency for certain applications and addresses some confidentiality concerns by keeping the network on-premises. Use cases like private 5G as a managed service [54, 57] and inter-operator roaming [16] require flexibility in distributing network functions between core and edge. Private 5G is a concept encompassing the provision of tailor-made network applications and services to private businesses and third-party providers [66]. Due to the high costs of deploying a standalone private 5G network on-premises, businesses can benefit from 5G services provided by mobile network operators through managed service providers. However, confidentiality concerns may inevitably emerge due to potential conflict of interest and the risk of data (e.g., configs, network topology [54]) leakage to third party providers. Our work addresses such concerns by leveraging confidentiality-preserving, anonymization, and pruning mechanisms.

Communication and Event Dependency Models. We review two existing models to capture communications and event dependencies that will later be applied in our method.

Communication Model. This model depicts the existence of communications between network functions or between an NF and another network element (e.g., gateway). Communication models are usually represented as directed graphs including two types of nodes, namely, internal nodes representing NFs belonging to the local cluster, and external nodes representing nodes belonging to other clusters (which can be either the source or destination of communications with internal nodes). The presence of an edge between two nodes indicates a directed network communication between those two nodes, while no edge means no communication exists between them. Communication models capture the connectivity dependency between entities (e.g., reachability) and can be used to detect network breaches and anomalous communications [17, 64]. As an example, Fig. 2a shows the communication model (as a graph) for NFs inside one cluster. For instance, SMF initiates a connection with several other NFs, while it only accepts a connection initiated by AMF.

Event Dependency Model. An event dependency model depicts the occurrence dependency (e.g., precedence or succession) between two events. Event dependency models are represented with directed graphs where nodes represent events and directed edges represent dependencies between events. Event dependency models are well studied (e.g., [25, 29, 34]) and used in different security solutions, such as event prediction [37, 45] and anomaly detection [11]. The models can be constructed through gathering application events (e.g., from logs, API calls, and messages) and establishing dependencies between such events based on different metrics (e.g., frequency and closeness) or techniques (e.g., Bayesian learning, LSTM, and *n*-gram). Fig. 2b depicts an excerpt of event dependency model from a 5G core network. For instance, the event "AUSF-HandleUeAuthPostRequest" depends on the event "AMF-Handle-RegistrationRequest", with the latter further depending on "AMF-HandleInitiateUEMessage", where UE is the mobile user equipment.


UDR
NRF
PCF
UPF
Mongo DB
SMF
AUSF
WebUI
NSSF
AMF
UDM
AMF-Handle Initial Context Setup Response
AUSF-Handle UeAuthPostRequest
AMF-Handle RegistrationComplete
AMF-Handle ULNASTransport
AMF-Handle RegistrationRequest
AMF-Handle RegistrationUpdating
AMF-Handle InitialUEMessage
NSSF-Handle NSSelectionGet

(a) Communication model. **(b) Event dependency model.**

Figure 2: Excerpts of communication and event dependency models of a 5G core network

Threat Model. Our *in-scope threats* include security breaches and attacks that involve multiple clusters (i.e., cross-cluster threats). Such breaches and attacks may originate from misconfigurations, user mistakes, or exploits of vulnerabilities by either insiders or external attackers. We assume the breaches and attacks may be identified based on the communications or event dependency models using existing security solutions including (but not limited to) offline security auditing, runtime security monitoring, and security impact prediction (as demonstrated in Section 8). Like many existing confidentiality-preserving solutions [9, 32, 72], we assume honest-but-curious providers who follow the proposed methodology. Finally, the edge clusters may be hosted by third parties and hence are assumed to be concerned about potential leakage of confidential information to each other via their communications with the core.

Out-of-scope threats include breaches or attacks that are contained inside a single cluster (as these can be tackled using existing solutions [3, 11, 17, 37, 64]) or those that cannot be identified based on the communication or event dependency models (e.g., container or OS-level attacks). We do not consider malicious providers who deviate from our methodology. Finally, we assume the integrity of our solution itself, Kubernetes, and the underlying infrastructure (including the data sources, such as network traffic and application logs, used to build our models), and hence attacks that can tamper with these are out-of-scope.

3 CCSM

This section presents an overview of the CCSM approach.

Overview. Fig. 3 illustrates an overview of the CCSM approach to build cross-cluster security models for edge-core environments involving multiple Kubernetes clusters. The inputs to CCSM are network traffic and event logs collected from a multi-cluster environment. CCSM comprises two phases as follows. First, the *Building Local Models* phase is to create local models for each cluster including all the edges and core by following Steps 1 to 4. (1) CCSM constructs local models for both communications and event dependencies from network traffic and event logs, respectively. (2) It extends the local models by identifying external nodes that interacts with the core. (3) It prunes the other nodes in the local models that do not interact with the core. (4) It anonymizes all the sensitive information in the models (e.g., IP addresses, network function names) to protect the confidentiality of an edge (typically owned by different tenants) and share on a "need to know" principle. Second, the *Building Global Models* phase is to combine local models to encompass a global view of the security posture across all clusters by following Steps 5 and 6. (5) CCSM constructs the global model for both communication and event dependencies by aggregating local models for all edges and the core. (6) It prunes the global model from the edges' perspectives to enable the creation of edge-specific views that are sent back to respective edges. We further detail both phases in Sections 4 and 5.

Example. Fig. 3 shows a simplified example with the outputs of CCSM. (1) The *constructing local model* step forms the communication model with the IP addresses for network functions (`2.2.2.2` for the NF `AMF`) as the nodes and their communication as the edges as well as the event dependency model with the event names as the nodes (`registration`) and their transitions as the edges; where internal nodes are indicated as unfilled and external nodes as filled. (2) The *extending* step identifies external nodes belonging to the core on communication local models (marked as horizontally stripped) and identifies the nodes on event dependency local models that have dependencies with at least one node on core's local model (depicted as diagonally stripped) along with indicating the direction of these dependencies. (3) The *pruning* step removes the other external node (marked as filled black) and bottom middle (unfilled) node from the communication model, as well as left bottom three (unfilled) nodes from the event model, as none of them are directly connected to the core. (4) The *anonymization* step anonymizes the NF name (`AMF`), IP address (`2.2.2.2`), and event name (`registration`) to (`7d6e...`), (`192.168.26.5`) and (`892f...`), respectively, using confidentiality-preserving algorithms (e.g., Crypto-PAn [58]). (5) The *constructing global model* step aggregates three local models for Edge 1, Edge 2 and the core (where nodes are indicated using no-pattern, diagonal strips, horizontal strips, respectively). (6) The final step prepares the specific views for Edge 1 and Edge 2, while pruning the non-reachable part of a global model from that specific edge. We further elaborate on this in Examples 1-4.

Security Applications. CCSM enables several security applications. First, it allows individual clusters to detect and predict security issues in them that would not be completely possible using only local models. Second, edge-specific views offer the possibility to audit other (potentially untrusted) tenants of the cloud environments. Third, our global model and edge-specific views can help administrators represent and assess the state of complex, distributed cloud environments. We detail the use cases of CCSM in Section 8.

4 CCSM FOR COMMUNICATIONS

This section presents our methodology for building cross-cluster security models for communication (the upper part of Fig. 3).

4.1 Building Local Communication Models

We detail how CCSM constructs a local communication model.

Constructing. To build communication models, our approach requires to capture network traffic between NFs. To later aggregate the local models, our approach relies on the external communications that each model shares with other clusters. CCSM identifies two types of communications: between two internal IP addresses

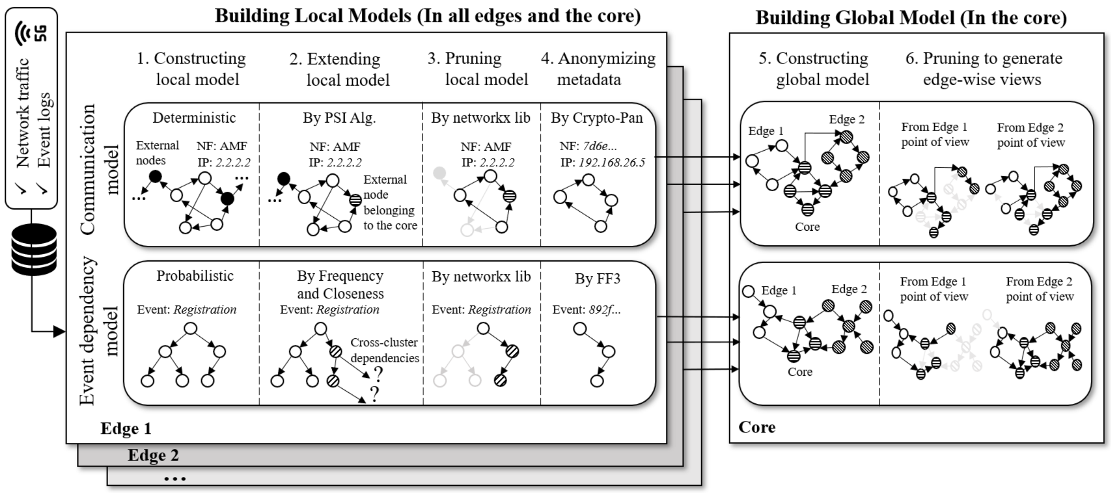


Figure 3: Overview of the CCSM approach

(*internal*) and between an external and an internal IP addresses (*external*). A single communication entry comprises the IP addresses of the source and the destination of a communication, and it also stores the name of the corresponding NF for internal IP addresses. For communication models, this process is deterministic, as we construct a directed graph with one node per unique IP involved in the communication, and a directed edge between two given nodes based on the existence of at least one communication record between source and destination nodes in the network traces. Nodes are labeled with the associated IP addresses and the name of the corresponding NFs.

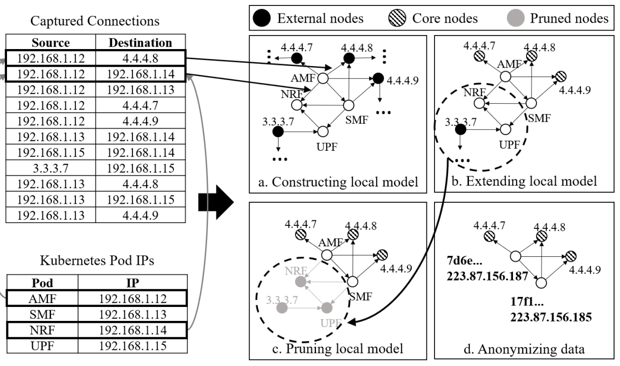
Extending. This step is to distinguish those external nodes shared with the core cluster from other external nodes (e.g., public IP addresses) in a confidential manner. To that end, our approach is to identify external nodes in the edges' local model that belong to the core cluster (and virtually *extend* the reach of the edges' local models to the core). The edge and the core clusters share their respective external communications with each other and identify those in common without disclosing any sensitive data. More precisely, to verify common connections between a given edge cluster and the core cluster without disclosing any sensitive data (i.e., the NF name or IP address), we utilize the private set intersection (PSI) algorithm [53], a cryptographic method for secure multiparty computation [10]. PSI enables two entities to compute and find the shared elements between their sets while keeping the non-shared items confidential. Once common communications are identified, each edge tags the corresponding external nodes as belonging to the core cluster.

Pruning. Local models can have a sheer size and therefore would require intensive computational resources during the following steps. Additionally, the network architecture that can be inferred from the model may still contain potentially sensitive information that individual clusters might be reluctant to share. To address both these challenges, our idea is to prune the local models and only preserve nodes and edges that are relevant to the global model by removing nodes (and corresponding edges) that have no path from/to an external nodes belonging to the core cluster. Our pruning technique allows to greatly reduce the size of the local models (as evaluated in Section 7) while preserving confidentiality by removing nodes that are irrelevant for the security analysis.

Anonymizing. While disclosure of sensitive information of an edge is partially ensured during *extending* and *pruning* steps, anonymization aims at further decorrelating the edge cluster specific information from its communication model. Therefore, our approach anonymizes any sensitive data before sending the local model for aggregation in the core cluster. In fact, the NF names and IP addresses (which are sensitive data as they can be traced back to that particular edge cluster) are anonymized using a format-preserving encryption (FPE) algorithm [5]. Particularly, for IP addresses, we leverage Crypto-PAn [69, 70], an FPE algorithm specifically designed for anonymizing IPs while preserving their subnet structures (which might be useful for later investigations). Table 1 in Appendix B shows several examples of applying Crypto-PAn on IP addresses.

Note that Crypto-PAn is utilized in this work as an example. Any alternative, such as Mohammady et al. [46], can also be used to overcome any potential concerns (e.g., fingerprinting and injection [7, 8, 71]). More generally, format-preserving encryption can also be used for other attributes based on necessity, type of application, and customer's requirements. For other applications, CCSM can anonymize different types of attributes (e.g., timestamps, owner, namespace, etc.) using more general methods such as a generalized framework which are provided by Xie et al. [68]. Similarly, the FF3 standard [20] is used to anonymize NF names.

Example 1. Figure 4 depicts an example of local communication model building. First, in Fig. 4.a, CCSM constructs the initial version of the local model (as a graph) based on the collected data, which represents communications between four NFs (e.g., AMF, NRF, SMF, and UPF) and four external IPs which are shown in the *Captured Connections* and *Kubernetes Pod IPs* tables. The first two entries of the *Captured Connections* table are used to create an edge for the external communication between the AMF (`192.168.1.12`) and an external IP (`4.4.4.8`), and for the internal communication between AMF and NRF. Then, in Fig. 4.b, CCSM identifies three of those external IPs (`4.4.4.7`, `4.4.4.8`, and `4.4.4.9`) as belonging to the core cluster (through the use of PSI, not depicted in the figure) and marks them accordingly. In Fig. 4.c, the nodes corresponding to NRF, UPF, and external IP `3.3.3.7` are pruned, because none of those can reach to or can be reached from any nodes in the core. Conversely, SMF and AMF can reach the core IPs, therefore these nodes are kept. Finally, in Fig. 4.d, the name and IP addresses of remaining internal nodes, such as AMF, are replaced with the anonymized name, `7d6e...`, and anonymized IP address, `223.87.156.187`, by using FF3 and Cryto-PAn, respectively. Algorithm 1 in Appendix C describes the pseudo-code of this phase.



Captured Connections

Source	Destination
192.168.1.12	4.4.4.8
192.168.1.12	192.168.1.14
192.168.1.12	192.168.1.13
192.168.1.12	4.4.4.7
192.168.1.12	4.4.4.9
192.168.1.13	192.168.1.14
192.168.1.15	192.168.1.14
3.3.3.7	192.168.1.15
192.168.1.13	4.4.4.8
192.168.1.13	192.168.1.15
192.168.1.13	4.4.4.9

Kubernetes Pod IPs

Pod	IP
AMF	192.168.1.12
SMF	192.168.1.13
NRF	192.168.1.14
UPF	192.168.1.15

Figure 4: Example of the building local models phase

4.2 Building Global Communication Model

We detail how CCSM constructs global communication models.

Constructing the Global Model. In this step, we collect and aggregate all local models (edges' and core's) based on their shared nodes (previously identified in the *extending* step). The core cluster takes local models (including its own) as graphs and computes their mathematical union. It is made possible to find common nodes between graphs since the external nodes (belonging to the core) are not anonymized in the previous step. The obtained global communication model reflects how the edge clusters potentially interact with each other through the shared nodes in the core.

Pruning to Generate Edge-specific Views. Once the global model is built, it captures all cross-cluster relationships. However, following the "need to know" principle, each edge cluster is only entitled to view the fraction of the global communication model that is related to it. Therefore, we apply multiple times the same pruning process as in the *pruning local model* step to the global model while considering a different edge's point of view at a time. Specifically, for each edge, we create a restricted view of the global communication model by removing any node that cannot be reached from, or cannot reach an internal node of the original local model. This ensures that each edge cluster receives only the relevant information needed to improve their local security view, based on sharing the least required information without unnecessary overhead. Finally, once the edge-specific views are constructed, each view is securely transmitted to the corresponding edge cluster (by leveraging existing methods, e.g., TLS).

Example 2. Figure 5 depicts an example of building a global model from local models of Edge 1, Edge 2, and the core. First, in Fig. 5.e, those three local models are aggregated into a global model based on common external nodes previously identified (i.e., AUSF, SMF and UPF for Edge 1; and UDM and AUSF for Edge 2). Then, in Fig. 5.f, the global model is pruned to provide edge-specific views. For the Edge 1 specific view, its internal nodes AMF and SMF are related to the node (anonymized) `566e...` in Edge 2 through the Core's SMF and UDM. Additionally, they are related to the node `4ed4...` by transitivity. However, the last node of Edge 2, `a807...`, is excluded as it has no relationship with Edge 1 (i.e., no reachability to/from Edge 1). Similar logic can be applied from the point of view of Edge 2. Algorithm 2 in Appendix D presents the pseudo-code of this phase.

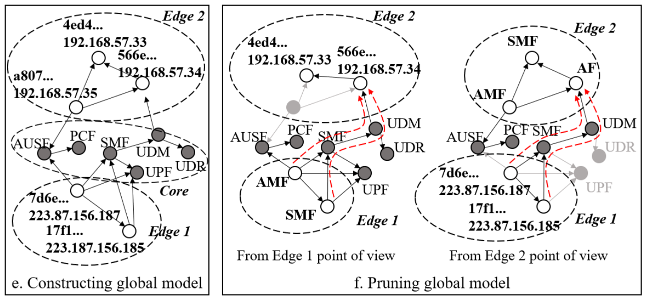


Figure 5: Example output of the global model building phase

5 CCSM FOR EVENT DEPENDENCIES

This section presents our methodology for building cross-cluster security models for event dependencies (the lower part of Fig. 3).

5.1 Building Local Event Dependency Models

During this step, CCSM builds local dependency models capturing the relationship between local events in edges and core.

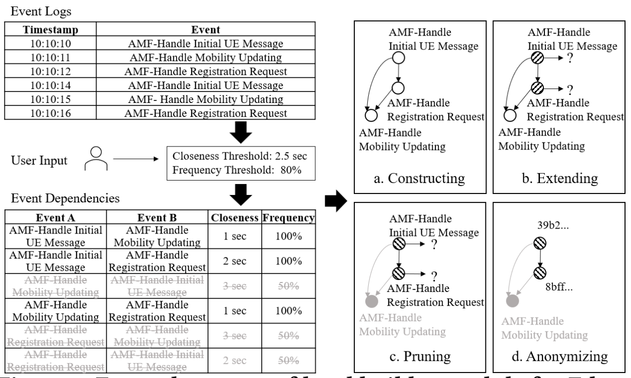
Identifying Event Dependencies. Unlike communication models, where dependency relationships can be assessed based on the presence or not of a network communication, the dependency between two application events cannot be identified in a similar manner. Instead, our key idea is to rely on two properties related to ordered pairs of event instances, namely, *temporal closeness* and *occurrence frequency*. We compute those properties for each ordered pair of event instances in the cluster and compare them to their respective user-defined thresholds to determine whether they have a dependency relationship or not between their corresponding event types. We present here our methodology to identify event dependencies, which we leverage to construct the local event dependency models, and then extend them to find the cross-cluster event dependencies used for constructing the global model. More formally, given a set of collected event types E = $\{E_1, E_2, \cdots, E_n\}$ where n is the number of different identified events types. Let (e_p, e_q) be an ordered pair of event instances, where event e_p happens at time t_p, and $t_q > t_p$ means event e_p happens before event e_q. We define event dependency between two event types in a given ordered pair (E_i, E_j) as a Boolean function that captures the existence (or not) of a dependency between those event types if the following holds: $\frac{|\{(e_p,e_q)|(t_q-t_p)\leq T_c\}|}{NumberOf(e_p)} > T_f$, where $T_c \geq 0$ is a user-defined threshold for the temporal closeness, which is defined as $t_q - t_p$ for any two instances of events, namely, e_p of type E_i and e_q of type E_j. $T_f \in [0, 1]$ is the user-defined threshold for the occurrence frequency of ordered pair of events instances (e_p, e_q) when its closeness is less than or equal to T_c. This condition expresses that the occurrence frequency should be strictly greater than T_f. Note that |_| means the set size and $NumberOf(e_p)$ counts the total number of occurrences of the event instances e_p of type E_i independently of the instances of event e_q. Thus, only a subset of the possible dependencies of the form $E_i \longrightarrow E_j$ will be considered in the event dependency model, which satisfy the closeness threshold and frequency threshold chosen by the user. We show how the choice of these thresholds impacts the dependency model generated by CCSM and provide guidelines on how they can be chosen depending on the use case in Section 7.

Constructing. To build the event dependency model, we first collect 5G event logs at the application level. This collection process occurs at the NF level using Kubernetes' Pods and containers logging. Then, event logs from all 5G applications are subsequently merged and sorted based on timestamps. Following this, the logs are parsed to extract the event name (usually in the form of `<NF name>-<associated 5G procedure>`. Afterwards, CCSM constructs a local model based on the two aforementioned parameters (which will be referred as *closeness* and *frequency*, respectively, for the rest of the paper) to find dependency edges. More specifically, for any given ordered pair of event types (E_i, E_j), we count the number of times that an instance of event type E_i is followed by an instance of event type E_j within a time interval smaller or equal to a predefined closeness threshold. Subsequently, the frequency of event instances of type E_j after event instances of type E_i satisfying the temporal closeness condition, is computed and compared with the frequency threshold. If the computed value is larger than the threshold the dependency is considered, otherwise it is discarded. This probabilistic approach provides flexibility to users in defining the dependency relationship between event types and allows them to customize these two parameters to identify the best values for their specific application (e.g., anomaly detection applications might benefit from smaller frequency thresholds, as anomalies happen in a non-regular and local manner). This effect is later evaluated in Section 7.

Extending. To extend the scope of the local models and further consider cross-cluster dependencies, we apply our dependency identification method between ordered pairs of events from different models. First, CCSM anonymizes event names, and then shares the events logs with the core cluster. Note that event timestamps are not anonymized as their exact values are needed by the core cluster to establish further event dependencies. Unlike for communication models where PSI is used to identify common nodes in a confidentiality-preserving manner, the same reasoning cannot be applied since cross-cluster event dependencies have to be assessed based on parameters, particularly occurrence frequency of ordered pairs consisting of both edge's events and core's events. Therefore, it is necessary that edge clusters share their entire anonymized event logs with the core for a centralized processing. At the core cluster, CCSM employs our dependency identification approach described above for each ordered pair of event types, E_i and E_j, belonging each to a different cluster (edge or core). Following this, the core cluster returns to the edge the list of edge's events that are involved in cross-cluster dependencies along with indicating the direction of dependencies (i.e., events that have a dependency with at least one event on the core cluster, striped diagonally in Fig 6).

Pruning. This step is to remove non-essential nodes and edges while ensuring that important events (i.e., that play a role in the global model) are retained by pruning nodes that have no path from or to any nodes identified during the *extending* step.

Anonymizing. The final step in the *building local model* phase is to anonymize the local model before sending it for aggregation in the core cluster. In this step, we utilize FF3 [42], a format-preserving


Event Logs

Timestamp	Event
10:10:10	AMF-Handle Initial UE Message
10:10:11	AMF-Handle Mobility Updating
10:10:12	AMF-Handle Registration Request
10:10:14	AMF-Handle Initial UE Message
10:10:15	AMF- Handle Mobility Updating
10:10:16	AMF-Handle Registration Request

User Input

Closeness Threshold: 2.5 sec
Frequency Threshold: 80%

Event Dependencies

Event A	Event B	Closeness	Frequency
AMF-Handle Initial UE Message	AMF-Handle Mobility Updating	1 sec	100%
AMF-Handle Initial UE Message	AMF-Handle Registration Request	2 sec	100%
~~AMF-Handle Mobility Updating~~	~~AMF-Handle Initial UE Message~~	~~3 sec~~	~~50%~~
AMF-Handle Mobility Updating	AMF-Handle Registration Request	1 sec	100%
~~AMF-Handle Registration Request~~	~~AMF-Handle Mobility Updating~~	~~3 sec~~	~~50%~~
~~AMF-Handle Registration Request~~	~~AMF-Handle Initial UE Message~~	2 sec	~~50%~~

AMF-Handle Initial UE Message
AMF-Handle Registration Request
AMF-Handle Mobility Updating
a. Constructing
AMF-Handle Initial UE Message
?
?
AMF-Handle Registration Request
AMF-Handle Mobility Updating
b. Extending
AMF-Handle Initial UE Message
?
?
AMF-Handle Registration Request
AMF-Handle Mobility Updating
c. Pruning
39b2...
8bff...
AMF-Handle Mobility Updating
d. Anonymizing

Figure 6: Example output of local builder module for Edge 1

encryption algorithm, to anonymize event names in the model, similarly to how we anonymize NF names for the communication model.

Example 3. Figure 6 depicts an example of building a local event dependency model. Closeness and frequency threshold values are 2.5 s and 80%, respectively. First, in Fig. 6.a, CCSM constructs a local model as a graph by finding dependencies based on event logs and processing tables which are shown in the figure. For instance, there is no edge from *AMF-Handle Registration Request* to *AMF-Handle Initial UE Message* in the event dependency local model since its frequency (50%) is lower than the frequency threshold, even though it meets the closeness threshold. Then, in Fig. 6.b, the graph is extended by tagging nodes which have cross-cluster dependency (shown with striped patterns). For instance, *AMF-Handle Initial UE Message* and *AMF-Handle Registration Request* are found to have dependencies with the core cluster based on the closeness and frequency they have with certain core events (not shown in this figure). After that, in Fig. 6.c, the extended model is pruned by removing irrelevant nodes and edges which are not essential for the analysis. For instance, the node *AMF-Handle Mobility Updating* is pruned as it cannot reach any node in the core, and cannot be reached from any node in the core. Finally, in Fig. 6.d, event names are anonymized by using FF3 in order to address confidentiality concerns. The *AMF-Handle Registration Request* is anonymized (to `39b2...`).

5.2 Building Global Event Dependency Model

We detail how CCSM constructs global event dependency models.

Constructing the Global Model. In this step, we integrate all local models (edges' and core's) based on the cross-cluster dependencies previously identified. Since cross-cluster event dependencies are identified in the core only, a consequence of this is that edge clusters are not aware of the core-side of their dependencies. Based on the cross-cluster dependencies identified during the *extending* step, the core is able to connect the prune version of local models with its own model, effectively constructing a global model.

Pruning to Generate Edge-specific Views. Once the global model is built, we apply our pruning method from each edge's point of view. For each edge, we create a restricted view of the global model by removing any node that cannot be reached from (or cannot reach) an internal node of the edge's original local model. At the end of this step, each edge receives only the necessary information to improve their local security (i.e., each edge sees a specific view from global model). This approach to generate and send a specific view to each edge addresses confidentiality concerns of individual clusters.

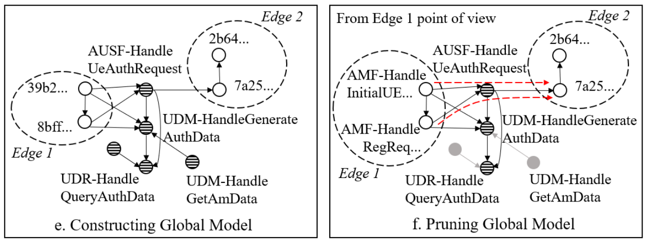


Figure 7: Example output of global builder module on core

Example 4. Figure 7 depicts an example of building a global event dependency model from three local models (of Edge 1, Edge 2, and the core). In Fig. 7.e, the model of Edge 1 (left), Edge 2 (top right), and the core (middle) are aggregated. To that end, CCSM uses the closeness and frequency of events to find cross-cluster dependencies between edge and core nodes. From this step, it finds three new dependencies between events of Edge 1 and the core: (starting with) `39b2...`→*AUSF-HandleUeAuth...*, `39b2...` →*UDM-HandleGenerate ...*, and `8bff...`→*UDM-HandleGenerate...* (the corresponding event logs are not shown). Similarly, it finds one additional event dependency between *AUSF-HandleUeAuth...* (in the core) and `7a25...` (in Edge 2). Then, in step Fig. 7.f, CCSM prunes the global model from the point of view of Edge 1, by removing core events that cannot be reached from (or cannot reach) any event from Edge 1 (for instance, *UDM-HandleGetAmData*). As a result, the obtained model view shows a potential event dependency from events in Edge 1 (`39b2...` and `8bff...`) to an event in Edge 2 (`7a25...`), indicating a possible breach which might be examined by experts.

6 IMPLEMENTATION

This section details the implementation of CCSM. We deploy a multi-cluster containerized 5G core network based on Towards5GS-Helm [1], an open-source initiative based on Free5GC [27]. We use Helm charts to automate this deployment on Kubernetes. We use UERANSIM [30] to simulate the 5G Radio Access Network and the UEs to generate events and traffic in the 5G core (e.g., registration and de-registration of multiple UEs). We implement CCSM following the architecture depicted in Fig. 16 (shown in Appendix E, due to space constraint), which also shows the technologies employed to realize each component.

To build the communication models, we first collect the control plane traffic on each cluster by deploying a DaemonSet of Pods running TShark [14] to capture network traces on each node. We enable the Kubernetes `hostNetwork` option and run the Pod as `privileged` to collect the traffic on the host interface. A server on the master node receives all traffic logs as .pcap files and extracts each connection (i.e., source and destination IP). Then, we map each source and destination IP address to its corresponding Kubernetes Pod by querying the Kubernetes API. If no mapping can be found, we label the IP as external (i.e., one of the communicants is outside of that specified cluster network). On the other hand, to build the event dependency graphs, we first collect all logs using the Kubernetes API (`kubectl logs` command) directly from the master node and then extract the event logs. Then, we aggregate and sort all event logs by timestamp (regardless of their node or function of origin), and use a custom Python script to find the event dependencies, as described in Section 5.

We use the Python library Pandas [67] to process the tabular data and generate the initial communication or event dependency models. Then, we first set up an open-source implementation of the PSI algorithm based on [53] as a server on the core cluster. For the communication graph, we identify common IP addresses between each edge's communication and the core's NFs using a custom script. To prune local models, we employ the *Pandas* [67] and *Networkx* [40] Python libraries. We anonymize IP addresses using an implementation of Crypto-PAn [58]. On the other hand, other attributes like the event names or NF names are anonymized with FF3 [42], a NIST-approved format-preserving encryption algorithm. On the core cluster, we develop custom scripts employing PSI, the *Networkx*, and the *Pandas* Python libraries in a similar manner to aggregate local models, build a global model, and finally send the pruned global model to each edge cluster.

During CCSM implementation, we encounter several challenges that are described in Appendix F, due to space constraint.

7 EVALUATION

This section details the experimental evaluation of CCSM.

7.1 Experimental Settings and Datasets

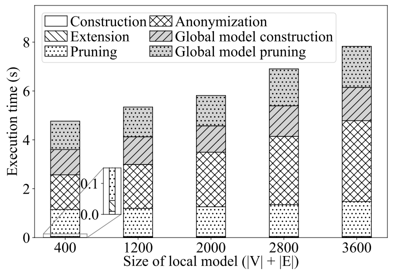
Experimental Settings. CCSM is deployed across two Kubernetes clusters, each composed of one master node and two worker nodes. Each node is a virtual machine with 4 vCPUs and 8GB memory, running Ubuntu 20.04. We use VirtualBox as hypervisor, and create an internal network for each cluster and an internal network for the communication between clusters (backhaul). We deploy Kubernetes v1.23.14 on top of Containerd v1.4.6 using `Kubeadm`. The physical hardware of our cloud is composed of one server with 2x Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz and 128GB of DDR4-2933.

Datasets. We collect data from our testbed. To generate several communication models, we use a sample communication model built based on the collected logs (as shown in Fig. 2) as a seed to generate configurations with multiple clusters, each with different NF-splitting between edge and core. For the event dependency models, we use several UEs to trigger 5G procedures (e.g., registration and mobility) in our multi-cluster 5G core, then we collect the Free5GC application logs by running `kubectl logs` command for each Pod. We collect 65,946 event logs of 61 unique events over two days in a 5G environment composed of 10 edge and one core clusters.

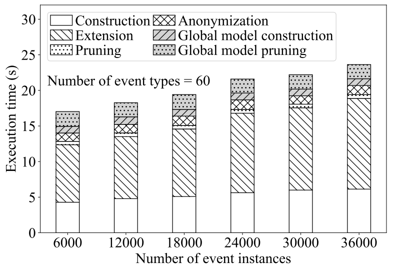
7.2 Experimental Results

In the following, we present the evaluation results of CCSM.

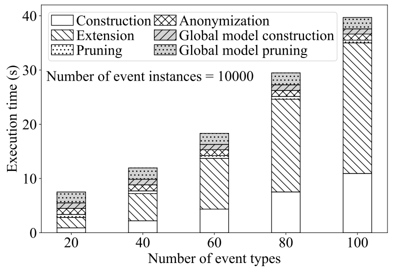
Execution Time. First, we measure the execution time needed by CCSM to perform each step under representative parameter values. For those experiments, we only consider one core cluster and one edge cluster. Fig. 8a shows the time required for building communication models depending on the size of the local models (i.e., the sum of edges and vertices: |V| + |E|), ranging from 400 to 3600 (before pruning). Overall, the execution time of CCSM including *local model building* and *global model building* appears to grow linearly. Some specific steps (e.g., pruning) show negligible impact



(a) **Time to build communication models as a function of the size of local models**



(b) **Time to build event dependency models as a function of the total number of events**



(c) **Time to build event dependency models as a function of the number of unique events**

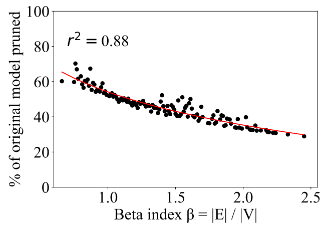
Figure 8: Evaluation of execution time

by the size of the local models, while others (e.g., anonymization, performed using a third party tool) are more dependent on that size. Overall, the execution time of all steps grows almost linearly.

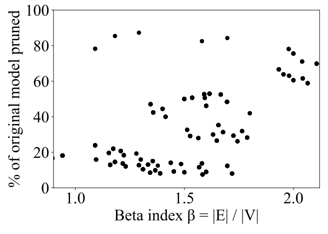
Fig. 8b and 8c show the total time required by CCSM to build both local and global event dependency models while varying the number of event instances and event types, respectively. In the case of event dependency models, as the size of the model is not representative of different types of clusters (i.e., a large cluster can generate very few events, or a small cluster can generate a vast amount of events), we instead study the impact of two other parameters specific to the event dependency models, namely, the number of event instances and the number of unique event types. Fig. 8b presents the execution time of each step for event instances ranging from 6,000 to 36,000, while the number of unique event type is fixed to 60 (which is representative of the number of event types collected in our testbed for a common scenario). It can be observed that the trend is almost linear, and the local model extension step is the most time-consuming as it involves searching for cross-cluster dependencies between the edge and core. Additionally, Fig. 8c shows the effect of increasing the number of unique event types, while the number of event instances is fixed to 10,000. The execution time of CCSM appears quadratic, ranging from around 8 s to almost 40 s, when the number of event types grows from 20 to 100. This is expected since each new unique event type might form dependency with all the existing event types, leading to a quadratic growth in the execution time.

Since our approach is offline in nature (i.e., the global communication and event dependency models are only to be built periodically), we conclude that using CCSM in a real-world environment is efficient (most execution times are in seconds) and scalable (linear or quadratic growth). Also, CCSM is by design scalable to more edge clusters since the steps for building local models can be executed in parallel between those edge clusters, while our experiences show that the execution time for building global models only shows a negligible increase in the number of edge clusters, thus ensuring our approach is practical even for large edge-core environments.

Effectiveness of Pruning. In this set of experiments, we evaluate the effectiveness of pruning techniques utilized to remove the irrelevant nodes and edges. To that end, we measure and compare the size of models before and after pruning. Fig. 9 depicts the size of pruned models as a percentage of the size of original models vs. the local models' *beta* index [19] (the *beta* index is a measure of a graph's connectivity and is calculated as the ratio between the number of edges (|E|) and the number of nodes (|V|) in the graph). Fig. 9a depicts our results for communication model. As *beta* indices increase from 0.7 to 2.5, the effectiveness of pruning diminishes exponentially. For a low *beta* index value, the local models are so sparse that a significant number of nodes and edges could be irrelevant to the global model and can thus be pruned. Therefore, the size of the pruned models averages only 50% of their original sizes when *beta* index is less than 1. On the other hand, as the *beta* index increases, most of the nodes could become connected so pruning is less effective. It is to be noted that this is not a limitation of our pruning method, but rather an expected consequence of the changing models. As a general guideline, one can skip the pruning steps when dealing with a highly connected model (e.g., *beta* index > 2). On the other hand, as shown in Fig. 9b, the results for event dependency models show no clear dependency on the level of connectivity of the models, while our approach can prune between 10% and 60% of the local model. The lack of a trend can be explained by the fact that event dependencies are determined by the types of events and thus are less regular than communications (e.g., as demonstrated in Figure 2).



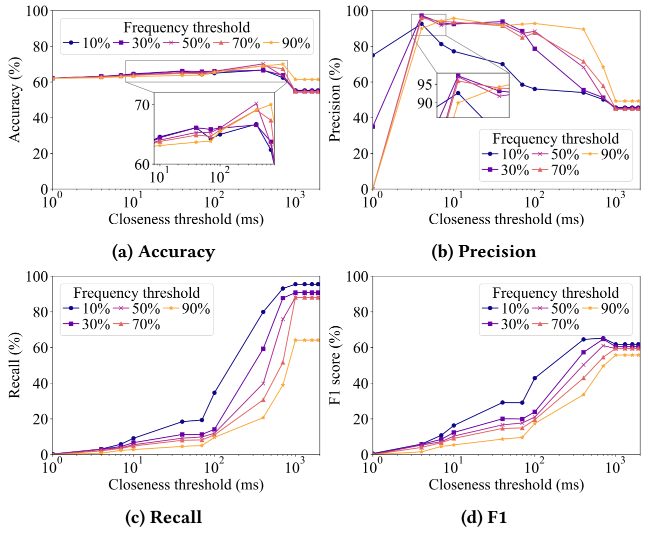
(a) **Communication models**



(b) **Event dependency models**

Figure 9: Evaluation of pruning effectiveness

Performance of Event Dependency Models. In this set of experiments, we measure the accuracy of our event dependency models. Because event dependency models are built in a non-deterministic manner (i.e., based on temporal closeness and frequency of observation), we assess the quality of our method by comparing our event dependency models to the ground truth event dependency in a 5G core environment (i.e., following the 3GPP standard [22]). Our goal is to measure with what accuracy CCSM can reconstruct the event dependencies across multiple clusters using closeness and frequency as a way to extend and aggregate local models (as detailed in Section 5). Fig. 10 shows the accuracy, precision, recall, and F1 score of global models built by CCSM under different closeness and frequency threshold values, where the measurements are for cross-cluster event dependencies identified by CCSM. We apply different metrics since these may be important for different applications.



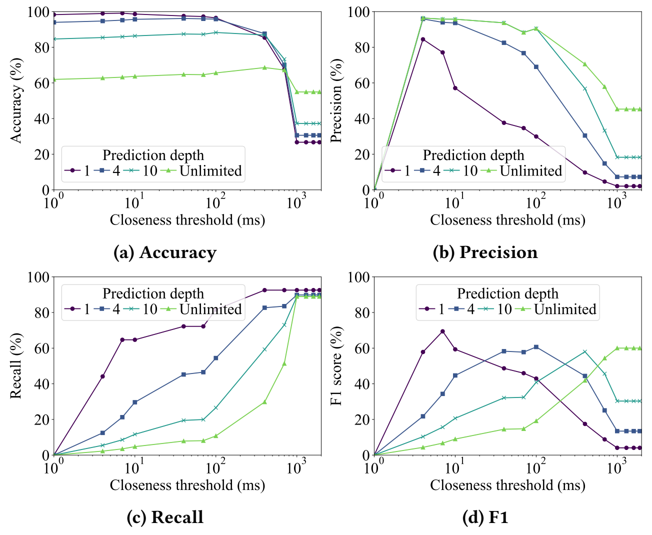
(a) Accuracy **(b) Precision**

(c) Recall **(d) F1**

Figure 10: Performance of CCSM event dependency models for various closeness T_c and frequency T_f thresholds

The results show that the overall accuracy of CCSM reaches a peak at slightly over 70% with a frequency threshold of 50% and a closeness threshold of 500 ms (i.e., two events are considered dependent if and only if they happen within less than 500 ms at least 50% of the time). Decreasing the frequency threshold overall results in a lower precision (i.e., more false positive as more events would be considered as dependent) but higher recall (i.e., less false negatives). Conversely, increasing the frequency threshold results in higher precision of our approach, but lower recall. We observe similar behaviour when varying the closeness threshold. Increasing it past 1,000 ms significantly drops the accuracy of our solution since Free5GC procedures (e.g., UE registration/de-registration) typically happen within one second according to our observations. Considering lower closeness thresholds generally increases the precision but decreases the recall, and vice versa. We conclude that optimal closeness and frequency thresholds may be identified based on the specific needs of different security applications. We further discuss the choice of such threshold values in Section 9.

Event Prediction Evaluation. We further extend the evaluation of event dependency models to the application of predicting future events [37, 44]. Specifically, we fix the frequency threshold of our solution at 70%, and evaluate the performance of our learned models for predicting events at depth=1 (i.e., the immediate next event), depth=4 (i.e, the next four events), depth=10, and without a prediction depth limit. Fig. 11 shows that the peak accuracy of 99% is reached when predicting the immediate next event with a low closeness threshold (10 ms). Under similar conditions, our solution correctly predicts the four next events with 96% accuracy, 10 next events with 88% accuracy, and finally all future events with 68% accuracy (similarly to Fig. 10). We can observe that CCSM has better precision at predicting events to unlimited depths (i.e., events that happen after an arbitrary number of other events) at the cost of recall. Those results again emphasize the importance of customizing the threshold values for specific use cases. For instance, users interested in predicting immediate next events with high accuracy could use a closeness threshold between 10 ms and 100 ms, whereas predicting all future events can use a threshold higher than 1,000 ms.



(a) Accuracy **(b) Precision**

(c) Recall **(d) F1**

Figure 11: Performance of event prediction using CCSM compared to ground truth for various prediction depths (T_f = 70%)

8 USE CASES

This section describes three cross-cluster use cases of CCSM.

Use Case 1: Cross-Cluster Security Verification. This first use case is to verify if an application deployed across multiple clusters complies with security policies [33]. Well-known examples of such security policies are network isolation [74], communication pattern [17, 64], and/or event-based properties [33, 37]. The top part of Fig. 12 illustrates an example of two 5G network slices deployed across edge and core clusters whose isolation is being compromised by a security attack [26]. For instance, the attacker first exploits a vulnerability (e.g., CVE-2021-41794) to gain unauthorized initial access to the UPF2 within the edge cluster and then performs a lateral movement (e.g., container escape using CVE-2022-0492). The attacker eventually breaches the boundaries between clusters due to compromised network slice isolation, resulting in a new network communication highlighted by a red arrow between UPF2 in the edge cluster and AUSF in the core cluster. In this scenario, the end-to-end isolation between 5G network slices needs to be verified across the edge and core clusters. The local communication models are shown on both sides of the deployed network. However, those local models by themselves do not allow to detect the breach. By constructing the global model (shown at the bottom of the figure) from the available logs, CCSM allows the identification of such unwanted communication between the two clusters (network isolation) or other anomalous communication patterns. Therein, the unexpected communication between the UPF2 at the edge cluster and the AUSF at the core cluster will be identifiable by the added arrow in red between those two NFs.

Use Case 2: Cross-Cluster Security Impact Prediction. The second use case refers to predicting the potential impact of a planned update, performed locally in some parts of the system (software or hardware), on the security of the whole multi-cluster system.

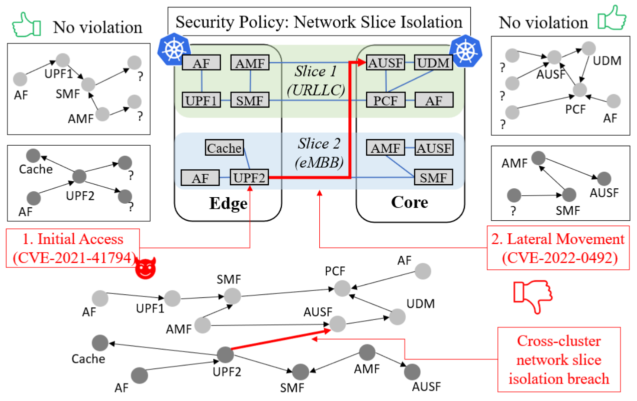

No violation
Security Policy: Network Slice Isolation
No violation
AF
AMF
Slice 1 (URLLC)
AUSF
UDM
UPF1
SMF
PCF
AF
Cache
Slice 2 (eMBB)
AMF
AUSF
AF
UPF2
SMF
Edge
Core
1. Initial Access (CVE-2021-41794)
2. Lateral Movement (CVE-2022-0492)
Cross-cluster network slice isolation breach

Figure 12: Use Case 1 - Cross-cluster security verification

Fig. 13 shows how CCSM can help in predicting the security impact due to a configuration change to one part of the system causing perturbation to the whole system using an example inspired from recent telecom operators outage [24, 49] where an update due to maintenance caused a failure in network routing. The middle part of Fig. 13 shows the deployed network functions (NFs) in the edges and the core, where only the virtual router NF (highlighted in red) has to be updated. The right part of the figure shows the local communication models for the edges (identical for all edges) and the core before the update is performed. By comparing the global model before the update (not shown here) and the global model (shown in the left side) after the update (performed in a test environment) generated by CCSM, we can see that the model captures the filtering/routing disruptions caused by the update (shown as dashed red arrows denoting the absence of communication between the impacted NFs). Particularly, even though the routing functionality is maintained among NFs inside the core, this update has broken the communications between the AUSF in the core at one side and the AMFs in edges at the other side. This is due to the fact that the AMF requires communicating with the AUSF, which has become unreachable because of the update. In summary, as the global model obtained by CCSM (as an impact of those updates) diverges significantly from its prior state, our solution empowers administrators to extend existing forecasting and change impact analysis techniques to multiple Kubernetes clusters.

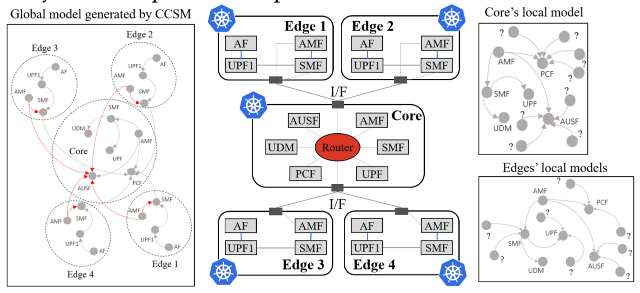


Figure 13: Use Case 2 - Cross-cluster impact prediction

Use Case 3: Cross-Cluster Anomaly/Attack Detection. Our third use case is to detect anomalies across multiple clusters that would otherwise go undetected. Fig. 14 depicts an example scenario involving a central cloud and two private 5G networks [57]. Specifically, the private cloud provides connectivity to on-site users, while

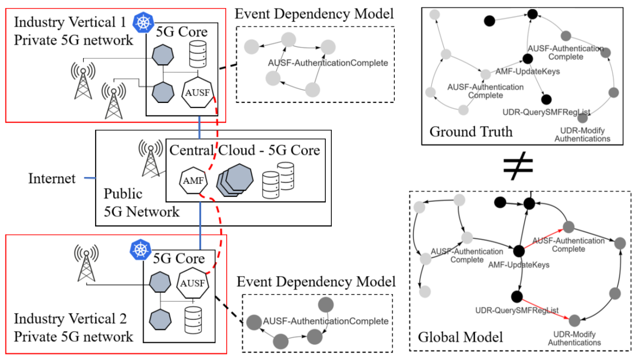


Figure 14: Use Case 3 - Cross-cluster anomaly detection

the central cloud (e.g., mobile network operator) provides roaming for private cloud users off-site. Following the authentication procedure in the private cloud, the authentication function (AUSF) of each private cloud transmits cryptographic keys to the management function (AMF) at the public cloud in order to secure the UE's communications while roaming. These extended authentication protocol (EAP)-based procedures are detailed in the 3GPP specifications [23] and can be represented as a *reference global model* depicting the expected behavior (shown at the top right corner). Similar to existing local anomaly detection models [17, 64], CCSM can be used to detect global, cross-cluster anomalies (e.g., a potential attack on the core from a user connected to an edge). For example, the red edges show that the `AMF-UpdateKeys` and `UDR-QuerySMFRegList` events are unexpectedly related to events in the second private cloud, potentially indicating a leak of data from the public 5G network to the private 5G network 2 (e.g., cryptographic keys). Although event dependency models generated by CCSM are built in a non-deterministic manner and might result in inaccuracies (see Section 7), they remain well-suited for anomaly detection tasks where the main goal is to raise alerts for further examination by security analysts.

9 DISCUSSION

Adapting CCSM to Other Orchestrators. While CCSM is implemented for and integrated with Kubernetes, our methodology is platform-agnostic. In particular, CCSM can be ported to other container orchestrators (e.g., Docker Swarm [18] and OpenShift [51]) with minimal engineering effort, as it only requires access to network logs, Pod names and application logs in each cluster. In particular, Docker Swarm's "Swarms" and OpenShift's "Clusters" can be used to deploy containers in different environments. IP addresses and network logs can be collected similarly using Wireshark [14], and application logs can be accessed using the orchestrators' respective command line interface.

Model Completeness and Correctness. CCSM 's communication models depend on the correctness and the completeness of the collected network traces. Our model can be further enriched with external information (e.g., NAT rules, VPN traces) to tackle inaccuracies on those traces (e.g., caused by address translation systems such as VPN, NAT, and proxies [4, 35]). The correctness of our event dependency models depends on the precision of the log timestamps and quality of the time synchronisation in the cluster, and choosing the right thresholds can greatly improve the accuracy of constructed models (Section 7). Finally, both models can be updated periodically to reflect changing communication patterns and/or changes in the architectures (e.g., addition of an application, deletion of a network connection) which could affect the accuracy of those models; which can be performed in a relatively short time (see Fig. 8).

Choosing a Threshold Value. Our evaluation results (Section 7) show how the choices of threshold values have a consequence on the size and the accuracy of generated models, and should be adapted depending on the use case. Overall, large closeness thresholds and small frequency thresholds result in larger models. When choosing threshold values, users should also take into account the application and its requirements. For instance, predictive approaches (e.g., event prediction [37] or security impact prediction presented in Use Case 2) might benefit more from a larger closeness threshold and a higher recall (i.e., less false negatives) as the increased rate of false positives (i.e., wrong predictions that only add a slight delay) would be acceptable. On the other hand, anomaly detection applications (e.g., Use Case 3) might benefit from a higher precision (i.e., less false positives) to reduce alert fatigue. This corresponds to a smaller closeness and larger frequency to solely capture events that appear frequently together at close time intervals.

10 RELATED WORK

There exist many solutions addressing different aspects of cloud and 5G security including security verification, attack detection, and impact prediction. The authors in [17] detect security breaches in software-defined networks based on network topology and forwarding rule. SFC-Checker [64] validates service function chaining correctness based on network topology, ensuring correct forwarding behavior. These solutions are not designed for a multi-cluster environment with confidentiality concerns, and our solution enables their applicability to such environments. In [74], the authors study different approaches for detecting network service anomalies in NFV environments. Our solution can potentially support the detection of the same topology anomalies, but additionally the anomalies happening over multiple clusters. There exist several proactive security compliance verification works (e.g., [6, 37, 43, 44, 52]) for OpenStack clouds. For instance, Weatherman [6] and Congress [52] verify security policies in clouds using graph-based and Datalog-based models, respectively. *LeaPS* [44] and *Proactivizer* [45] are proactive security verification solutions for cloud environments based on event dependency models, while *ProSPEC* [37] adopts a similar approach but works for container environments. *WARP* [3] applies the dependency model to proactively mitigate attacks in Kubernetes clusters. In [73], the authors propose an advanced approach to mine correlated events, which can increase the reliability of event dependency models. The authors in [25] propose a probabilistic approach to match and discover correlated events, while the authors in [29, 34] follow a frequency-based metric to assess the dependency of events. While all those works are limited to events and dependency models in a single cloud or cluster, our solution can enable their extension to multi-cluster environments.

There exist solutions for distributed and collaborative security applications. For instance, in [12], the authors propose a framework to predict advanced persistent threats (APT) in a distributed 5G environment while ensuring confidentiality. In contrast to such application-specific solutions, our work provides a methodology for building global models to enable different security applications. Federated learning (FL) is another popular approach to collaboratively train a global machine learning model, which has found many security applications recently. For instance, the authors in [65] propose a FL approach to detect network attacks in 5G distributed systems using packet traffic analysis. Mothukuri et al. [47] introduce a FL-based anomaly detection approach, enhancing IoT security through collaborative learning techniques. Su et al. [60] apply edge-based FL in a smart grid environment to share the private energy data of users with minimum communication overhead while ensuring data confidentiality. Rasha et al. [55] study the application of FL in smart cities (e.g., transportation, healthcare, and communication) for improving security and confidentiality. Although FL might train a global machine learning model without sharing user data, it usually requires multiple iterations which may impact the network performance, and it might also suffer from distributed data and model poisoning [59]. While the high-level objectives of our work are similar to FL (i.e., building global models while preserving confidentiality), the main difference lies in their inputs and outputs, i.e., FL is designed to build machine learning (ML) models from raw data, whereas our solution aims to piece together existing local models (which may or may not have been constructed using ML) into a global model (which again may not be an ML model in nature).

11 CONCLUSION

In this paper, we presented CCSM that builds cross-cluster security models to enable various security analyses, while preserving confidentiality for each cluster. We instantiated our solution based on both the communication model and event dependency model of edge-core environments. We showed the application of our solution for different use cases. We implemented and integrated our solution into a multi-cluster 5G core testbed. Our experimental results demonstrate the performance of CCSM for various use cases. However, CCSM currently has a few limitations which will be addressed in our future work. For instance, the potential of our method to generalize beyond those two models (i.e., communication and event dependency) will be explored in the future. Also, we will extend our models by adding more details, e.g., the number of connections and amount of data exchanged over a communication link, or the probabilities of event dependencies, to cover more security applications.

REFERENCES

[1] Abderaouf Khichane. 2023. Towards5Gs. https://github.com/Orange-OpenSource/towards5gs-helm
[2] Ijaz Ahmad, Tanesh Kumar, Madhusanka Liyanage, Jude Okwuibe, Mika Ylianttila, and Andrei Gurtov. 2017. 5G Security: Analysis of Threats and Solutions. In *IEEE CSCN*.
[3] Sima Bagheri, Hugo Kermabon-Bobinnec, Suryadipta Majumdar, Yosr Jarraya, Lingyu Wang, and Makan Pourzandi. 2023. Warping the Defence Timeline: Non-disruptive Proactive Attack Mitigation for Kubernetes Clusters. In *IEEE ICC*.
[4] Cataldo Basile, Daniele Canavese, Christian Pitscheider, Antonio Lioy, and Fulvio Valenza. 2017. Assessing Network Authorization Policies via Reachability Analysis. *Computers & Electrical Engineering* 64 (2017), 110–131.
[5] Mihir Bellare, Thomas Ristenpart, Phillip Rogaway, and Till Stegers. 2009. Format-Preserving Encryption. In *SAC*.
[6] Sören Bleikertz, Carsten Vogel, Thomas Groß, and Sebastian Mödersheim. 2015. Proactive Security Analysis of Changes in Virtualized Infrastructures. In *ACSAC*.
[7] Tønnes Brekne and André Årnes. 2005. Circumventing IP-address Pseudonymization. In *IASTED CCN*.
[8] Tønnes Brekne, André Årnes, and Arne Øslebø. 2006. Anonymization of IP Traffic Monitoring Data: Attacks on Two Prefix-Preserving Anonymization Schemes and some Proposed Remedies. In *PET*.

[9] Justin Brickell and Vitaly Shmatikov. 2005. Privacy-preserving Graph Algorithms in the Semi-honest Model. In *ASIACRYPT*.
[10] Hao Chen, Kim Laine, and Peter Rindal. 2017. Fast Private Set Intersection from Homomorphic Encryption. In *ACM CCS*.
[11] Long Cheng, Ke Tian, Danfeng Daphne Yao, Lui Sha, and Raheem A Beyah. 2019. Checking is Believing: Event-Aware Program Anomaly Detection in Cyber-Physical Systems. *IEEE TDSC* 18, 2 (2019), 825–842.
[12] Xiang Cheng, Qian Luo, Ye Pan, Zitong Li, Jiale Zhang, and Bing Chen. 2021. Predicting the APT for Cyber Situation Comprehension in 5G-Enabled IoT Scenarios Based on Differentially Private Federated Learning. *Security and Communication Networks* (2021), 1–14.
[13] CNCF. 2022. Survey. https://cncf.io/reports/cncf-annual-survey-2022
[14] Gerald Combs. 2023. TShark. https://wireshark.org/docs/man-pages/tshark.html
[15] Marius Corici, Pousali Chakraborty, and Thomas Magedanz. 2021. A Study of 5G Edge-Central Core Network Split Options. *MDPI Network* 1, 3 (2021), 354–368.
[16] Marius Corici, Pousali Chakraborty, Thomas Magedanz, Andre S. Gomes, Luis Cordeiro, and Kashif Mahmood. 2021. 5G Non-Public-Networks (NPN) Roaming Architecture. In *International Conference on Network of the Future*.
[17] Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, and Vijay Mann. 2015. SPHINX: Detecting Security Attacks in Software-Defined Networks. In *NDSS*.
[18] Docker. 2023. Docker Swarm. https://docs.docker.com/engine/swarm
[19] César Ducruet and Jean-Paul Rodrigue. 2013. Graph Theory: Measures and Indices. *The Geography of Transport Systems* (2013).
[20] Morris Dworkin. 2016. *Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption*. Technical Report. NIST Special Publication.
[21] Ericsson. . 5G Cloud Infrastructure. https://ericsson.com/en/cloud-infrastructure
[22] ETSI. 2017. *3GPP Technical Specification (TS) 23.502 version 16.7.0 Release 16 - System Architecture for the 5G System*. Technical Report.
[23] ETSI. 2020. *3GPP Technical Specification (TS) 33.501 version 17.0.0 - Security Architecture and Procedures for 5G System*. Technical Report.
[24] Pete Evans. 2022. Rogers says services mostly restored after daylong outage left millions offline. https://cbc.ca/news/business/rogers-outage-cell-mobile-wifi-1.6514373
[25] Diogo R Ferreira and Daniel Gillblad. 2009. Discovering Process Models From Unlabelled Event Logs. In *BPM*.
[26] Enduring Security Framework. 2022. *ESF Potential Threats to 5G Network Slicing*. Technical Report. NSA, CISA, ODNI.
[27] Free5GC. 2023. https://free5gc.org
[28] Free5GC. 2023. Free5GC GitHub Repository. https://github.com/free5gc/free5gc
[29] Yu Gao, Shaoxu Song, Xiaochen Zhu, Jianmin Wang, Xiang Lian, and Lei Zou. 2018. Matching Heterogeneous Event Data. *IEEE TKDE* 30, 11 (2018), 2157–2170.
[30] Ali Güngör. 2023. UERANSIM. https://github.com/aligungr/UERANSIM
[31] Bin Han, Antonio DeDomenico, Ghina Dandachi, Anastasios Drosou, Dimitrios Tzovaras, Roberto Querio, Fabrizio Moggio, Omer Bulakci, and Hans D Schotten. 2018. Admission and Congestion Control for 5G Network Slicing. In *IEEE CSCN*.
[32] Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, Jiajun Du, and Dawu Gu. 2022. Shuffle-based Private Set Union: Faster and more Secure. In *USENIX Security*.
[33] José María Jorquera Valero, Pedro Miguel Sánchez Sánchez, Alexios Lekidis, Javier Fernandez Hidalgo, Manuel Gil Pérez, M Shuaib Siddiqui, Alberto Huertas Celdran, and Gregorio Martínez Pérez. 2022. Design of a Security and Trust Framework for 5G Multi-Domain Scenarios. *Journal of Network and Systems Management* 30, 1 (2022).
[34] Jaewoo Kang and Jeffrey F Naughton. 2003. On Schema Matching with Opaque Column Names and Data Values. In *ACM SIGMOD Conference*.
[35] Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Header Space Analysis: Static Checking for Networks. In *USENIX NSDI*.
[36] Sami Kekki et al. 2018. *MEC in 5G Networks*. Technical Report. ETSI.
[37] Hugo Kermabon-Bobinnec, Mahmood Gholipourchoubeh, Sima Bagheri, Suryadipta Majumdar, Yosr Jarraya, Makan Pourzandi, and Lingyu Wang. 2022. ProSPEC: Proactive Security Policy Enforcement for Containers. In *CODASPY*.
[38] Rabia Khan, Pardeep Kumar, Dushantha Nalin K Jayakody, and Madhusanka Liyanage. 2019. A Survey on Security and Privacy of 5G Technologies: Potential Solutions, Recent Advancements, and Future Directions. *IEEE COMST* 22, 1 (2019), 196–248.
[39] Kubernetes 2023. https://kubernetes.io
[40] Los Alamos National Laboratory. 2023. NetworkX. https://networkx.org/
[41] Mohammed Laroui, Boubakr Nour, Hassine Moungla, Moussa A Cherif, Hossam Afifi, and Mohsen Guizani. 2021. Edge and Fog Computing for IoT: A Survey on Current Research Activities & Future Directions. *Computer Communications* 180 (2021), 210–231.
[42] Schoening Consulting LLC. 2023. FF3. https://pypi.org/project/ff3
[43] Suryadipta Majumdar, Yosr Jarraya, Taous Madi, Amir Alimohammadifar, Makan Pourzandi, Lingyu Wang, and Mourad Debbabi. 2016. Proactive Verification of Security Compliance for Clouds through Pre-Computation: Application to OpenStack. In *ESORICS*.
[44] Suryadipta Majumdar, Yosr Jarraya, Momen Oqaily, Amir Alimohammadifar, Makan Pourzandi, Lingyu Wang, and Mourad Debbabi. 2017. LeaPS: Learning-Based Proactive Security Auditing for Clouds. In *ESORICS*.
[45] Suryadipta Majumdar, Azadeh Tabiban, Meisam Mohammady, Alaa Oqaily, Yosr Jarraya, Makan Pourzandi, Lingyu Wang, and Mourad Debbabi. 2019. Proactivizer: Transforming Existing Verification Tools into Efficient Solutions for Runtime Security Enforcement. In *ESORICS*.
[46] Meisam Mohammady, Lingyu Wang, Yuan Hong, Habib Louafi, Makan Pourzandi, and Mourad Debbabi. 2018. Preserving both Privacy and Utility in Network Trace Anonymization. In *ACM CCS*.
[47] Viraaji Mothukuri, Prachi Khare, Reza M Parizi, Seyedamin Pouriyeh, Ali Dehghantanha, and Gautam Srivastava. 2021. Federated-Learning-based Anomaly Detection for IoT Security Attacks. *IEEE IoTJ* 9, 4 (2021), 2545–2554.
[48] Jin Nakazato, Makoto Nakamura, Tao Yu, Zongdian Li, Gia Khanh Tran, and Kei Sakaguchi. 2020. Design of MEC 5G Cellular Networks: Viewpoints from Telecom Operators and Backhaul Owners. In *IEEE ICC Workshops*.
[49] CBC News. 2021. Rogers says service starting to return after Canada-wide wireless outage. https://cbc.ca/news/business/rogers-outage-1.5992954
[50] Council of the European Union and European Parliament. 2016. General Data Protection Regulation (GDPR). *Official Journal of the European Union - L* 119 (2016), 1–88.
[51] OpenShift. 2023. OpenShift. https://docs.openshift.com/
[52] OpenStack. 2015. https://wiki.openstack.org/wiki/Congress
[53] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2014. Faster Private Set Intersection Based on {OT} Extension. In *USENIX Security*.
[54] Wint Yi Poe, Jose Ordonez-Lucena, and Kashif Mahmood. 2020. Provisioning Private 5G Networks by means of Network Slicing: Architectures and Challenges. In *IEEE ICC Workshops*.
[55] Al-Huthaifi Rasha, Tianrui Li, Wei Huang, Jin Gu, and Chongshou Li. 2023. Federated Learning in Smart Cities: Privacy and Security Survey. *Information Sciences* (2023).
[56] Dario Sabella et al. 2019. *Edge Computing: from Standard to Actual Infrastructure Deployment and Software Development*. Technical Report. ETSI.
[57] Peter Schneider, Christian Mannweiler, and Sylvaine Kerboeuf. 2018. Providing Strong 5G Mobile Network Slice Isolation for Highly Sensitive Third-party Services. In *IEEE WCNC*.
[58] Keiichi Shima. 2015. Crypto-PAn. https://github.com/keiichishima/yacryptopan
[59] Mengkai Song, Zhibo Wang, Zhifei Zhang, Yang Song, Qian Wang, Ju Ren, and Hairong Qi. 2020. Analyzing User-Level Privacy Attack Against Federated Learning. *IEEE JSAC* 38, 10 (2020), 2430–2444.
[60] Zhou Su, Yuntao Wang, Tom H Luan, Ning Zhang, Feng Li, Tao Chen, and Hui Cao. 2021. Secure and Efficient Federated Learning for Smart Grid with Edge-Cloud Collaboration. *IEEE Transactions on Industrial Informatics* 18, 2 (2021), 1333–1344.
[61] Submariner 2023. https://submariner.io/
[62] Sari Sultan, Imtiaz Ahmad, and Tassos Dimitriou. 2019. Container Security: Issues, Challenges, and the Road Ahead. *IEEE Access* 7 (2019), 52976–52996.
[63] Nan Sun, Jun Zhang, Paul Rimba, Shang Gao, Leo Yu Zhang, and Yang Xiang. 2018. Data-Driven Cybersecurity Incident Prediction: A Survey. *IEEE COMST* 21, 2 (2018), 1744–1772.
[64] Brendan Tschaen, Ying Zhang, Theo Benson, Sujata Banerjee, Jeongkeun Lee, and Joon-Myung Kang. 2016. SFC-Checker: Checking the Correct Forwarding Behavior of Service Function Chaining. In *IEEE NFV-SDN*.
[65] Yunkai Wei, Sipei Zhou, Supeng Leng, Sabita Maharjan, and Yan Zhang. 2021. Federated Learning Empowered End-Edge-Cloud Cooperation for 5G HetNet Security. *IEEE Network* 35, 2 (2021), 88–94.
[66] Miaowen Wen, Qiang Li, Kyeong Jin Kim, David López-Pérez, Octavia A Dobre, H Vincent Poor, Petar Popovski, and Theodoros A Tsiftsis. 2021. Private 5G Networks: Concepts, Architectures, and Research Landscape. *IEEE JSTSP* 16, 1 (2021), 7–25.
[67] Wes McKinney. 2010. Data Structures for Statistical Computing in Python.
[68] Shangyu Xie, Meisam Mohammady, Han Wang, Lingyu Wang, Jaideep Vaidya, and Yuan Hong. 2021. A Generalized Framework for Preserving both Privacy and Utility in Data Outsourcing. *IEEE TKDE* 35, 1 (2021), 1–15.
[69] Jun Xu, Jinliang Fan, Mostafa Ammar, and Sue B Moon. 2001. On the Design and Performance of Prefix-Preserving IP Traffic Trace Anonymization. In *ACM SIGCOMM Workshop on Internet Measurement*.
[70] Jun Xu, Jinliang Fan, Mostafa H Ammar, and Sue B Moon. 2002. Prefix-Preserving IP Address Anonymization: Measurement-Based Security Evaluation and a New Cryptography-Based Scheme. In *IEEE ICNP*.
[71] Ting-Fang Yen, Xin Huang, Fabian Monrose, and Michael K Reiter. 2009. Browser Fingerprinting from Coarse Traffic Summaries: Techniques and Implications. In *DIMVA*.
[72] Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. 2023. Linear Private Set Union from {Multi-Query} Reverse Private Membership Test. In *USENIX Security*.
[73] Marc-Andre Zöller, Marcus Baum, and Marco F Huber. 2017. Framework for Mining Event Correlations and Time Lags in Large Event Sequences. In *IEEE International Conference on Industrial Informatics*.
[74] Moubarak Zoure, Toufik Ahmed, and Laurent Réveillère. 2022. Network Services Anomalies in NFV: Survey, Taxonomy, and Verification Methods. *IEEE TNSM* 19, 2 (2022), 1567–1584.

A EXAMPLE OF EDGE-CORE ARCHITECTURE

The Mobile Edge Computing (MEC) standard [36, 56] suggests placing the User Plane Function (UPF) at the edge and the control plane network functions (e.g., Access and Mobility Management Function (AMF) and Session Management Function (SMF)) on the core, which can improve the response time and performance. Fig. 15 depicts an edge-core architecture while deploying AMF and SMF on the edge next to the UPF [15] (as depicted in Fig. 15), while the core provides authentication, authorization, and other control plane functionalities. Such an approach can considerably reduce signaling between the edge and core.

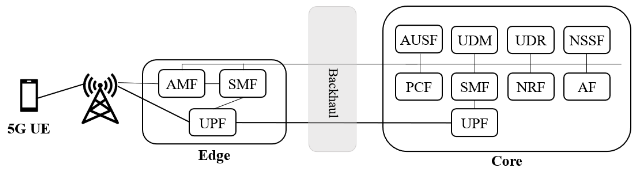


Figure 15: An example of 5G edge-core architecture [15]

B EXAMPLE OF FORMAT-PRESERVING ENCRYPTION

Table 1 shows several examples of IP addresses anonymized using Crypto-PAn in a prefix-preserving manner. For example, as first and second IPs in the table belong to the same subnet, after anonymization their corresponding anonymized IP addresses will belong to the same subnet again. In the same way, as the third one has 16 bits in common with two previous ones, after anonymization the corresponding anonymized IP address will have the same first 16 bits.

Table 1: An example of format-preserving encryption using Crypto-PAn

Original IP	IP anonymized by Crypto-PAn
192.168.1.13	***223.87.156.***185
192.168.1.14	***223.87.156.***187
192.168.5.23	***223.87.***155.187
192.130.3.54	***223.***125.128.117
10.10.10.25	29.21.233.153

C ALGORITHM FOR BUILDING LOCAL MODELS

Algorithm 1 shows our pseudo-code for building local models. There are four functions for the four steps of the *Building local models* phase. First, the *Constructing local model* function (Line 4) parses a `connections.csv` file and converts its content in a graph. Then, this graph is processed by the second function *Extending local model* (Line 14) to extract external connections and find common connections with the core cluster (or with other edges, for the core). To do so, the core and edges use the PSI algorithm to find common connections without endangering their own confidentiality. Third, the graph is processed by the *Pruning local model* function (Line 31) to check the reachability of nodes and edges between the edges and the core cluster. This function keeps those that are reachable and remove others. Finally, the *Anonymization local model* function (Line 47) applies anonymization techniques (e.g., PSI and FF3) to prepare the local model for aggregation (in the core cluster) while preserving confidentiality.

Algorithm 1 Building Local Model

```
 1: Input: Collected data $Connections.csv$ (SRC(IP, Name),DST(IP, Name))
 2: Output: Local model $G$
 3:
 4: function CONSTRUCTING_LOCAL_MODEL($Connections.csv$)
 5:     //Build local model from input file (Deterministic)
 6:     Temp Variable LocalModel($G = (E, V)$)
 7:     for each (SRC, DST) in $Connection.csv$ do
 8:         $G.V$.add(SRC)
 9:         $G.V$.add(DST)
10:         $G.E$.add(SRC, DST)
11:     Return $G$
12:
13: function EXTENDING_LOCAL_MODEL($G$)
14:     for edge e in $G.E$ do
15:         if e is an external connection then
16:             ExternalEdges.add(e)
17:     //Find common edges between edge cluster and core
18:     CommonEdges = PSI(ExternalEdges)
19:     for e in CommonEdges do
20:         if SRC is external IP then
21:             $G.V$(SRC).isCommon = 1
22:         else
23:             $G.V$(DST).isCommon = 1
24:     Return $G$
25:
26: function PRUNING_LOCAL_MODEL($G$)
27:     Temp Variable LocalModel($G' = (E, V)$)
28:     for node $n$ in $G.V$ do
29:         if $n$.isCommon == 1 then
30:             $G'.V$.add(ancestors(n))
31:             $G'.V$.add(descendants(n))
32:     for edge e in $G.E$ do
33:         if e.SRC is in $G'.V$ and e.DST is in $G'.V$ then
34:             $G'.E$.add(e)
35:     Return $G'$
36:
37: function ANONYMIZING_LOCAL_MODEL($G$)
38:     for node n in $G.V$ do
39:         n.SRC.IP = Crypto-PAn(n.SRC.IP)
40:         n.DST.IP = Crypto-PAn(n.DST.IP)
41:         n.SRC.Name = FF3(n.SRC.Name)
42:         n.DST.Name = FF3(n.DST.Name)
43:     Return $G$
```

D ALGORITHM FOR BUILDING GLOBAL MODEL

Algorithm 2 details the process of constructing the global model. First, the *Constructing global model* function (Line 4) merges all local models depending on their connections with the core. To do so, it constructs the set intersection between each edge's local model and its own local model using the PSI algorithm. Progressively, local models are merged with the core's local model to form a global model. Then, CCSM shares a version of the global model to each edge, containing only the information related to that edge. To that end, the *Pruning global model for edge* function (Line 21) removes all edges and nodes that are not reachable (or cannot reach) from (to) any node in that specific edge cluster.

E IMPLEMENTATION ARCHITECTURE

Figure 16 illustrates the architecture of CCSM that contains two major modules: local model bulder and global model builder. CCSM is implemented in a distributed manner where the *local model builder*

Algorithm 2 Building Global Model

```
Input: $LocalModels$, Edge number $j$
Output: Global model $G_i$ pruned for Edge number $j$

function CONSTRUCTING_GLOBAL_MODEL($LocalModels$)
    Temp Variable GlobalModel($G = (E, V)$)
    for Graph $G_i$ in $LocalModels$ do
        for edge e in $G_i^{'}.E$ do
            if e.SRC.isCommom == 1 or e.DST.isCommon == 1 then
                if verify_by_PSI_result(e) then
                    Continue
                else
                    Alarm for anomaly
        G.merge($G_i^{'}$)
    Return $G$

function PRUNING_GLOBAL_MODEL_FOR_EDGE($G$, $j$)
    Temp Variable GlobalModel($G^{'} = (E, V)$)
    for node $n$ in $G.V$ do
        if $n$ is a node from local model of edge $j$ then
            $G^{'}.V$.add(ancestors($n$))
            $G^{'}.V$.add(descendants($n$))
    for Edge $e$ in $G.E$ do
        if $e$.SRC is in $G^{'}.V$ and $e$.DST is in $G^{'}.V$ then
            $G^{'}.E$.add($e$)
    Return $G^{'}$
```

module instances are deployed on edges and core, and the *global model builder* module is deployed as a centralized manager on the core cluster.

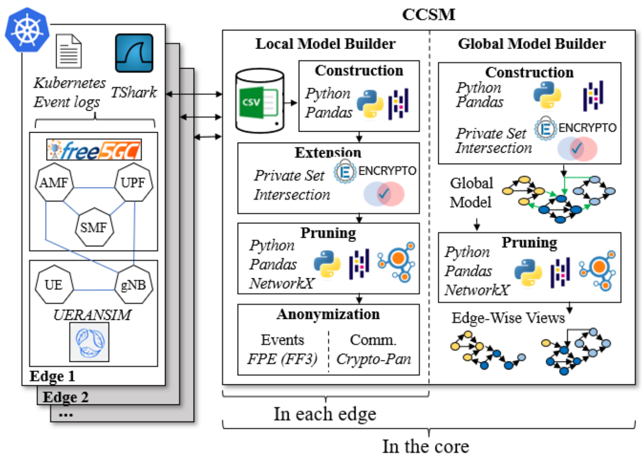


Figure 16: CCSM implementation architecture

F IMPLEMENTATION CHALLENGES

Timestamp Precision. To determine the relationships and the order between events, we rely on the timestamp measurements output by each application's logging function. However, using the default logs of Free5GC led to several inconsistencies as the default unit of timestamps precision is only at the second scale. Although this is not a challenge within a single NF's logs (as logs are written sequentially to standard output, in the correct order), reliably determining the order of events from different NFs is made impossible when all such events have the same timestamp. This behaviour is frequently seen since events in the 5G core may happen at a rapid pace (e.g., a complete UE registration can take less than one second and involve a dozen events). Free5GC does not offer runtime options to modify and increase the precision of timestamps in their logs, and therefore, to address such issues, we modify the source code of the NF applications (in Go) to instead print logs at the nanosecond scale (i.e., changing `TimestampFormat` from `Time.RFC3339` to `Time.RFC3339Nano`). This change has been officially merged into the vendor's official code base [28].

Multi-cluster 5G Core on Kubernetes. We implement CCSM in a realistic edge-core environment composed of multiple Kubernetes clusters. Similarly to a real MEC deployment [48], we connect those different clusters with a backhaul network and the corresponding network routing rules. To allow services and NFs from different clusters to communicate, we leverage Submariner [61] to connect overlay networks of different Kubernetes clusters. In particular, we deploy a service broker on the core cluster, a gateway on each cluster's master node, and a Submariner tunnel between each edge's gateway and the core cluster's gateway. For security reasons, we deploy our clusters in an internal network, using only private interfaces. However, another challenge arises as Submariner requires public interfaces (specifically, the interface for the default route) to automatically discover the gateway. To solve this issue, we temporary make Submariner use our private interfaces by changing the default route during the time of the discovery only. To avoid additional complexity, we ensure that Pods and Services CIDR do not overlap between each clusters. Additionally, we set up Free5GC to work across our multi-cluster Kubernetes environment by exporting all Kubernetes services involved in cross-cluster communications depending on the edge-core configuration user.