Implementation of OpenAPI Wireshark Dissectors to Validate SBI Messages of 5G Core Networks

Lukas Schauer*, Thorsten Horstmann*†, Steffen Druesedow‡, Michael Rademacher*†
*Hochschule Bonn Rhein-Sieg, Sankt Augustin, Germany, firstname.lastname@h-brs.de
†Fraunhofer FKIE, Bonn, Germany, firstname.lastname@fkie.fraunhofer.de
‡Telekom Deutschland GmbH, Berlin, Germany, firstname.lastname@telekom.de

***Abstract*—This paper introduces a novel Wireshark dissector designed to facilitate the analysis of Service-Based Interface (SBI) communication in 5G Core Networks. Our approach involves parsing the OpenAPI schemes provided by the 5G specification to automatically generate the dissector code. Our tool enables the validation of 5G Core Network traces to ensure compliance with the specifications. Through testing against three open-source 5G Core Network projects, we identified several issues where messages deviate from specification standards, highlighting the significance of our implementation in ensuring protocol conformity and network reliability.**

***Index Terms*—5G, SBI, OpenAPI, Protocol Validation, Wireshark**

I. Introduction

Mobile networks based on 5G have become a reality and different vendors provide commercial equipment for User Equipments (UEs), Radio Access Networks (RANs) and the 5G Core Network (5GC). Compared to previous generations of mobile networks, interoperability between hardware and software components from different vendors has become even more crucial. This demand is particularly evident due to the heavy usage of concepts like Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) and the resulting microservices, which on the one hand lead to agility and cost-efficiency, but on the other hand to strict requirements towards the specification of the used protocols [1]. In this work, we put these strict requirements to test because despite the fact that 5G is gaining more and more popularity, there is one thing nobody seems to care about: protocol validity.

There are reasons why protocol validity is a more crucial factor in 5G than it was for previous generations of mobile networks. The main driver for that change was a completely new concept for mobile core networks, away from vendor specific binary protocols and monolithic systems towards a well accepted and commonly known technology based on microservices and RESTful protocols based on OpenAPI. This change led to a large increase of different 5GC implementations, not just in the open-source communities - also new commercial players entered the market. Such a variety comes with advantages for Mobile Network Operators (MNOs) but also with challenges, especially since the 3rd Generation Partnership Project (3GPP) specifications are rather complex and sometimes leave space for interpretations. Ensuring that components from different vendors interoperate correctly with each other, but also with available emulators and testing environment can be a tedious task. It often means tough study of the specifications and manual validation of large traces, still leaving the human factor as final error source. Objective tooling for validation not only helps operators to ensure correct operation, it also helps implementers during the development cycle and eventually may contribute to a cleaner 5GC ecosystem overall.

The motivation for this work emerges from our practical experiments with different 5GC: **We found various inconsistencies and issues with different implementations for the 5G Service-Based Architecture (SBA)**. To further analyze these inconsistencies and issues, we started looking for tooling. However, to our surprise, we were unable to identify any. Specific tooling was seemingly available within the paid products of various vendors, but they obviously targeted only big MNO.

In the end we looked at our options: The 5G specifications contain ASN.1 definitions for the binary part of core network communication, which basically can already be validated by Wireshark in various ways, but they also contain OpenAPI specifications, which describe the HTTP/2 API that is being used as the primary way of communication within the SBA. We quickly found that OpenAPI is limited to the validation of single request-response-pairs. However, this also provides a chance for a fairly straight-forward process, which we present in this work. In particular, our contributions are the following:

1) **We describe our implementation of a validator for OpenAPI specifications.** We implemented this validator as a Wireshark dissector, since Wireshark was already being used by us and the scientific community in general.
2) **We used our validator to analyze a simple scenario** using various open source implementations of 5GC and identified various issues within these implementations.
3) Most importantly, **we published our implementation open source [2]**.

The rest of this work is structured as follows: In Section II we briefly provide the required background regarding OpenAPI, 5GC and Wireshark. Section III discusses different tools which are related to this work. The main part of this work is Section IV, where we describe our concept and implementation. To demonstrate the usage of our tool, Section V reports on the evaluation of three widely-used 5GC implementations. We provide a summary and future work in Section VI.

II. Background

The **OpenAPI** specification not only defines the API endpoints and their associated operations but also provides a structured way to describe the request and response data models [3]. This includes specifying the expected data formats, field types, and validation rules for the message payloads exchanged between Network Functions (NFs). For example, the data model for a Session Establishment Request sent to the Session Management Function (SMF) may include fields like the Subscription Permanent Identifier (SUPI) of type "string" with a regular expression pattern, and nested objects. Well-defined message structures are crucial in the 5GC, as they ensure consistent interpretation of the data across different network elements, facilitate automated validation, and reduce the likelihood of interoperability issues.

The 5G mobile networks have brought about a paradigm shift in the architecture and design of cellular core networks. One of the key innovations in 5G is the adoption of a SBA, where network functions are decomposed into modular services that communicate with each other through well-defined APIs [4]. This approach, known as the Service-Based Interface (SBI), promotes flexibility, scalability, and efficient resource utilization within the 5GC. The SBI in the 5GC comprises a set of web-based APIs that enable communication between various NFs, such as the Access & Mobility Management Function (AMF), Authentication Server Function (AUSF), Unified Data Management (UDM), and others. These APIs are defined using the OpenAPI Specification and follow the RESTful architectural style, employing standard HTTP methods (GET, POST, PUT, DELETE) and JSON data formats for request and response payloads [5].

Wireshark, the widely-used network protocol analyzer, plays an important role in troubleshooting and validating network communications. As a productive packet capturing and analysis tool, Wireshark provides a huge set of dissectors for nearly all common network protocols, to decode and inspect network packets and flows [6]. To enhance its protocol dissection capabilities, Wireshark can be extended through custom dissectors written in the Lua programming language [7]. These dissectors allow developers to define rules for parsing and displaying protocol data structures without the need to compile the code. Furthermore, Wireshark's ability to apply display filters, follow packet streams, and export captured data for further analysis makes it a solid tool for validating the correct implementation of the 5GC SBA.

III. Related Work

Validation of RESTful APIs has been comprehensively discussed in [8]. In their work, the authors propose the tool **QuickREST** which is built up-on the concept of property-based testing (PBT). The idea of PBT is to automatically generate input data and evaluate if the API reacts to this input according to the specification. This approach is significantly different from our concept. In this work, we passively monitor traffic and conduct the validation offline.

The **5G Trace Visualizer** by Deutsche Telekom is an open source toolkit that allows to visualize traces from a 5G core network [9]. It uses Wireshark in the background to decode the traces and combines its output with metadata from Kubernetes and/or OpenStack. The resulting data can be visualized using an open source UML diagram tool to generate message sequence charts or by using a Python based plotting library to draw various metrics. The Visualizer itself is scripted using iPython/Jupyter notebooks and contains various example scripts, e.g., for activity in the core network correlated to resource usage in a Kubernetes cluster. Unfortunately, no actual validation of data according to the OpenAPI descriptions is implemented.

APIClarity by Cisco is a tool designed for analyzing and validating APIs described using OpenAPI specifications. It offers a wide range of features, including automatic generation of OpenAPI documentation based on real traffic on existing APIs and fuzzing of API endpoints [10]. APIClarity is mainly designed to be used as proxy between an API and the rest of the world, but it also offers support for network taps that can passively capture API traffic. Unfortunately, APIClarity seems to be unsuited for the analyzing and validating traffic of 5GC, as it was designed primarily for small and simple APIs. Our attempts to import parts of the 5G OpenAPI documents were not successful.

IV. Concept and Implementation

Our approach is based on several software components interacting with each other. Figure 1 provides an overview of the plugin architecture. In the following, we will highlight several parts. We enumerated these parts in Figure 1 for a better readability throughout this section.

We decided to write a Wireshark ② plugin capable of parsing SBI messages, allowing for easy validation of specification conformity of core network components. The plugin itself is written in Lua, which makes it extremely portable. Only a few files need to be placed in the Wireshark plugins directory and it simply starts working, independent of CPU architecture, operating system or other user specific parameters.

The plugin is implemented as a post-dissector. This approach allows us to easily build upon the existing HTTP/2 dissection. Unfortunately, this means that currently no dissection of live-traffic is possible, all input data needs to have been captured previously, so a PCAP file ① is used as input.

The main plugin ③ begins by trying to find corresponding request and response headers as well as data. It uses Wiresharks TCP flow numbers and HTTP/2s internal stream IDs to find matching packets. For each pair, the path from the request headers is used to look up the corresponding part of the OpenAPI documents.

The 3GPP publishes and frequently updates their OpenAPI documents in a publicly available repository in the form of various yaml files [11]. We use these yaml files ④ as the ground truth for our tool. In theory, it would be possible to just put the corresponding yaml files in a directory for parsing by the plugin, but initial tests showed that this would take

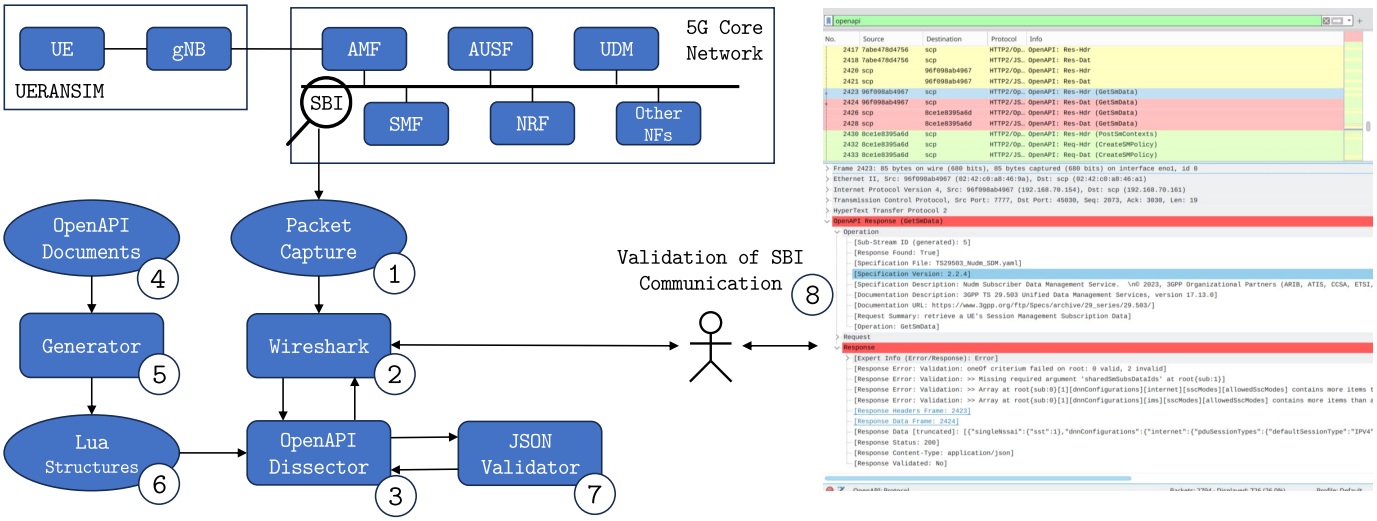

Fig. 1. Architecture of the implemented dissector and the evaluation scenario.

quite a considerable amount of time to find the corresponding specification for each request. To avoid having to wait for this work to be done on every launch of the dissector plugin, a Python based generator script ⑤ reorganizes path and component references in an easy-to-parse way. The script mainly creates a nested data structure ⑥, allowing easy access to required schema data indexed by request paths. These paths do not purely consist of static strings but allow for variables of various data types. The generator creates corresponding regular expressions for these complex path specifications. Thanks to the inclusion of the optimized regular expression library PCRE2 [12] inside of Wiresharks Lua interpreter, the patterns can quickly be resolved during runtime. For each request, the dissector searches this nested structure and provides the found schema data to the validation script.

When the connection establishment has not been captured (due to starting a capture in an already running core network) some headers might get lost due to HTTP/2s built-in compression. Our plugin tries to augment headers with best-effort estimations where possible. Some headers, e.g. Content-Type, are generated by looking at the format of the actual content, other headers are chosen from the OpenAPI documents.

After finding a matching path specification in the documents all headers, data and the path specifications get passed over to our validation script ⑦, which parses the provided JSON data and checks the given path specifications against it. For simple elements like strings and numbers the checks are rather simple. They basically consist of a list of conditions that might need to be matched (e.g. number in certain range, string matching a pattern, etc.) and return any found validation errors.

More complex elements like “objects” and “arrays” which contain other elements needed a bit more work, especially since the documentation of OpenAPI itself is very vague about some more specific behavior, e.g. when using negation. We approached this by implementing the behavior of our validation close to the corresponding sections we found in the 5G specifications OpenAPI documents.

The biggest challenge was adding support for `oneOf` / `anyOf` / `allOf` properties. They allow for an element at a specific location to be validated in various ways. They all take a list of various schemas which need to be validated against an element. The names imply how many of these schemas are allowed to validate for the given element. The `oneOf` property is special, because it allows for the additional definition of a discriminator value, which is then used to decide which of the given schemas the element gets validated against.

If any subelement validation fails the objects or arrays validation fails as well, so any validation errors are propagated back down to the JSON root element. Validation information and metadata is then added to the packet view ⑧ in Wireshark. With a simple click on a packet a user can see what document it was validated with, what operation it belongs to, have links to related packets (request/response, notifications/subscriptions), and especially see if it was detected as valid according to the OpenAPI document or if there have been any issues during validation.

V. Evaluation

To test our implementation we decided to evaluate current versions of three different open-source 5G Core Network projects: Open5GS (v2.7.0) [13], free5GC (v3.3.0) [14] and OpenAirInterface (v2.0.1) [15].

For this evaluation, we set up simple test networks, each with one of the available implementations and an instance of UERANSIM. UERANSIM is an open-source project that provides a software implementation of the RAN as well as the UE side of the 5G network [16]. It is designed to interoperate with any 5GC to enable end-to-end testing and validation of 5G network scenarios. To this end, it simulates the behavior of a gNodeB (gNB) and a UE, including the establishment of radio bearers, registration procedures, and exchanging control plane and user plane data with the 5GC. By omitting the

actual radio part, UERANSIM allows setting up functional 5G networks without the need for physical UE or gNB devices. Our test case for this work is the initialization of the 5G network, followed by a UE registration with corresponding Packet Data Unit (PDU) session establishment and ending with the simulated UEs deregistration. For each of the three evaluated 5GC projects, we performed this scenario and observed issues that our validator reported. Table I summarizes our results. In the following subsection, we provide detailed descriptions of the issues we found in the different implementations. This description should emphasize the complex nature of SBI messages and why a tool like the one presented in this work provides real benefit.

A. Open5GS

Open5GS is a prominent open-source implementation of the 5G Core Network, providing both 5G Standalone and Non-Standalone NFs. Developed primarily in the C programming language by a community of contributors, it is used by researchers and developers to experiment with and test 5G Core features and capabilities in a wide range of projects. During our tests, we found five notable issues regarding the OpenAPI specification correctness.

Service list in NF registration: During startup, some calls to the Network Repository Function (NRF) contain an empty `nfServiceList` during the `RegisterNFInstance` operation. According to the 3GPP Technical Specification (TS) 29.510, describing the NRF, the `nfServiceList` must contain at least one item (if present) [17].

Wrong version of UDR API: According to Release 17 of the 5G specification the UDR API should be v2 [18], but Open5GS still uses v1, which last seems to have been used in early versions of Release 15. We didn't further validate the correctness of these messages, as our validator currently has no support for multiple specification versions at the same time. Also the specifications for these early releases are not easily available since they are only included in the human-readable documentation of the 5G specification and are provided for information purposes only.

Creation of NRF subscriptions: A `subscriptionId` is generated by the NRF in the `CreateSubscription` messages, which identifies the corresponding subscription. The according NRF specification [17] describes this identifier to consist of the Mobile Country Code (MCC) and Mobile Network Code (MNC), followed by a generated part, but the ID used by Open5GS consists purely of a UUID.

Missing address information in resp. to NF registration: According to TS 29.510 [17] the response during an `RegisterNFInstance` operation should be an `NFProfile` description, which needs to contain at least one piece of addressing information (a fully qualified domain name, an IPv4 or an IPv6 adress), but the Open5GS NRF does not include this information in the response.

Handling of UEs session management subscription data: During the response to a `GetSmData` call the Open5GS UDM sends a `SessionManagementSubscriptionData` packet containing a `DnnConfiguration`. Inside the configuration `allowedSscModes` are defined for the IP Multimedia Subsystem (IMS) and the Internet service. According to the UDM specification, this list should contain a maximum of two entries [19], but in our example setup the Open5GS UDM returns a list containing three items.

B. free5GC

Free5GC is another notable actively developed open-source 5G Core Network implementation using the Go programming language. Free5GC's original goal was to provide academics with a platform to test and prototype 5G systems. Its feature set and open-source nature not only facilitate research and testing but also offer commercial value, particularly for deploying private 5G networks.

Invalid content-type for UEAuthenticationCtx: During POST requests containing `UEAuthenticationCtx` messages the content-type header is set to `application/json`, but the AUSF specification describes these messages as `application/3gppHal+json` [20]. While this probably would not cause any issues as it is the only expected data structure in this case, it still remains at least a minor violation of the provided specifications. The validator has a workaround for these special cases, where it basically ignores the given content type and looks up a content type from the specification. This only works for requests and responses with a single available content type, but it is better than simply ignoring the content of affected packets.

Wrong version of UDR and SDM API: Unfortunately, free5GC does not seem to specify the minor version of the used 5G specification version, so we assume the latest version of Release 15. Similar to Open5GS, it is still using UDR and various other APIs in v1, which are no longer in use since later versions of Release 15. As mentioned before, we have no machine-readable specifications for these older API versions and were unable to validate these messages accordingly.

Invalid ConfirmAuth messages: Instead of a valid `AuthEvent` message the response to `ConfirmAuth` messages is simply the value `null`. It also does not contain the expected location header to a created resource [18]. While the return data might not be important, especially with a correctly provided status code, this might still result in issues when used with different core network components.

Nulled content during SearchNFInstances call: During the response to a call to `SearchNFInstances` the `nfInstances` parameter is simply set to `null`. According to the specification [17], this should always be an array, but it might be empty if no instances have been found.

Invalid age values in UpdateSmContext messages: Inside the `UpdateSmContext` message an `ageOfLocationInformation` variable is set to a negative number, but according to the common data types specification for the SBI [21], this value needs to be in the range between 0 and 32767.

TABLE I
OVERVIEW ABOUT THE EVALUATION RESULTS. FIVE SELECTED KEY ISSUES IDENTIFIED PER OPEN-SOURCE 5GC IMPLEMENTATIONS.

Implementation	Issues				
Open5GS	Service list in NF registration	Wrong version of UDR API	Creation of NRF subscriptions	Missing address information in resp. to NF registration	Handling of UEs session management subscription data
free5GC	Invalid content-type for UEAuthenticationCtx	Wrong version of UDR and SDM API	Invalid ConfirmAuth messages	Nulled content during SearchNFInstances call	Invalid age values in UpdateSmContext messages
OpenAirInterface	Invalid NF registration	Wrong version of UDR API	Invalid NRF subscription condition	Invalid ConfirmAuth messages	Invalid SD encoding

C. OpenAirInterface

OpenAirInterface (OAI) is an open-source project consisting of both a new 5G Radio Access Network (RAN) and a Core Network implementation, running on general purpose x86 computing hardware and commercial off-the-shelf Software Defined Radio (SDR) cards. The code is developed by a worldwide industrial and academic community, governed under the OAI Software Alliance consortium.

Invalid NF registration: Similar to Open5GS, the most calls to the NRF contain an empty `nfServiceList` during the `RegisterNFInstance` operation, while they must contain at least one item (if present). In addition, the `SupiRange` type used by some operations is not encoded correctly. It contains the `pattern` attribute as well as `start` and `end` fields, which is not allowed. The specified values are also not in accordance with the specification [17].

Wrong version of UDR API: Similar to Open5GS and free5GC, OAI uses `v1` as endpoint API version for the `authentication-status` operation, while the specification requires `v2`. This shows a widespread misinterpretation of the specification across multiple projects.

Invalid NRF subscription condition: In the request to create a subscription to a specific NF instance in the NRF, the NF is determined according to different criteria specified by the `subscrCond` attribute of the `SubscriptionData` object type. This attribute is a composed schema that selects only one of the listed schemas under the keyword `oneOf`. Apparently, the encoding is implemented incorrectly by OAI. An additional mapping from `subscrCond` to `nfTypeCond` is inserted in the JSON structure to encode the `oneOf` concept. However, this does not match the expected OpenAPI specification [17], since the value of `subscrCond` becomes the composition of two objects rather than a single object. The issue is a consequence of a serious abandoned bug in the library "OpenAPI Generator" used by OAI for the OpenAPI client types de- and encoding. It is interesting to note, that the exact same problem was found in Ericsson's Network Core Test System by Davide Donato [22]. This emphasizes the complexity of the rules for encoding OpenAPI concepts, which makes manually writing code sensitive to inconsistencies and bugs.

Invalid ConfirmAuth messages: During the authentication process of an UE, the AUSF responses to the AMF the result of the `5g-aka-confirmation` operation as an `AuthResult` enumeration type [18]. The OAI implementation of the AUSF is using a simple boolean type instead of the specified enumeration values. This can lead to serious security problems if other AMF implementations are used that are unable to handle this incorrect encoding appropriately.

Invalid Slice Differentiator (SD) encoding: In several messages the SD is encoded incorrectly. The specification requires an exact 6-digit number to distinctive 5G network slices if they use the same Slice/Service Type (SST). However, OAI encodes the SD with variable length, which results in a minor specification violation [21].

VI. SUMMARY AND FUTURE WORK

This work presents our prototype for an OpenAPI Wireshark dissector to validate SBI messages of a 5GC. The implementation aspects are presented in detail in this work. In addition, we use a simple scenario to analyze messages from three different open-source 5GC implementations. Despite the simplicity of the scenario, we found various issues where the implementation differs from the protocol specification.

With the implementation of this tool, we want to contribute to a better implementation of 5G Core Networks. Additionally, we provide users, for example experts working at Mobile Network Operator or researchers, the possibility to better debug their network infrastructure. Lastly, if protocol validity is improved overall, this directly leads to more and better interoperability in the 5G ecosystem.

Although our prototype is working as expected and fully functional, we have already identified several aspects which can be improved. In the current state, certain components of the OpenAPI specification are implemented solely as minimal placeholders. Guessing the missing HTTP/2 header fields, resulting from stream compression, is based on a simple approach, which may lead to the selection of incorrect specifications for validation on rare occasions. Our main goal for future work is to add the possibility for validation of longer procedures, that consist of more than a simple request-response-pair. Our basic idea is to match against an ordered list of required operations to see if e.g. a NF registration and deregistration is done properly. This could be extended by matching service IDs, allowing for validation of multiple parallel sequences of operations, potentially also involving UEs in later steps.

REFERENCES

[1] J. B. Moreira, H. Mamede *et al.*, "Next generation of microservices for the 5g service-based architecture," *International Journal of Network Management*, vol. 30, no. 6, 2020.

[2] L. Schauer, "telekom/openapi-dissector: This repository contains experimental code for generating an openapi dissector for use within wireshark." https://github.com/telekom/OpenAPI-Dissector.

[3] "Openapi initiative," https://www.openapis.org/.

[4] G. Mayer, "Restful apis for the 5g service based architecture," *Journal of ICT Standardization*, vol. 6, no. 1-2, pp. 101–116, 2018.

[5] ETSI, "5G System; Principles and Guidelines for Services Definition," European Telecommunications Standards Institute (ETSI), TS 29.501 version 17.8.0, 01 2024.

[6] Wireshark Foundation, "Wireshark - Network Protocol Analyzer," https://www.wireshark.org/.

[7] "Lua Programming Language," https://www.lua.org/.

[8] S. Karlsson, A. Čaušević, and D. Sundmark, "Quickrest: Property-based test generation of openapi-described restful apis," in *2020 IEEE 13th ICST*. IEEE, 2020, pp. 131–141.

[9] Deutsche Telekom AG, "5g trace visualizer," https://github.com/telekom/5g-trace-visualizer.

[10] Cisco, "Apiclarity," https://www.apiclarity.io/.

[11] "All groups / 5g apis · gitlab," https://forge.3gpp.org/rep/all/5G_APIs.

[12] "Pcre2," https://github.com/PCRE2Project/pcre2.

[13] "open5gs.org," https://open5gs.org/.

[14] "free5gc," https://free5gc.org/.

[15] "Openairinterface – 5g software alliance for democratising wireless innovation," https://openairinterface.org/, (Accessed on 03/23/2024).

[16] A. Güngör, "Ueransim," https://github.com/aligungr/UERANSIM.

[17] 3GPP, "5G System; Network function repository services; Stage 3," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 29.510, 03 2024, version 17.13.0. [Online]. Available: http://www.3gpp.org/DynaReport/29510.htm

[18] 3GPP, "5G System; Unified Data Repository Services; Stage 3," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 29.504, 03 2024, version 17.13.0. [Online]. Available: http://www.3gpp.org/DynaReport/29504.htm

[19] 3GPP, "5G System; Unified Data Management Services; Stage 3," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 29.503, 01 2024, version 17.13.0. [Online]. Available: https://www.3gpp.org/dynareport/29503.htm

[20] 3GPP, "5G System; Authentication Server Services; Stage 3," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 29.509, 03 2024, version 17.10.0. [Online]. Available: http://www.3gpp.org/DynaReport/29509.htm

[21] 3GPP, "5G System; Common Data Types for Service Based Interfaces; Stage 3," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 29.571, 01 2024, version 17.10.0. [Online]. Available: http://www.3gpp.org/DynaReport/29571.htm

[22] D. Donato, "Using openapi 3 specifications of the 5g core to generate validators in erlang," Master's thesis, Chalmers University of Technology, 2019.