INTERNATIONAL SCIENTIFIC JOURNAL OF ENGINEERING AND MANAGEMENT (ISJEM) ISSN: 2583-6129
VOLUME: 05 ISSUE: 01 | JAN – 2026 DOI: 10.55041/ISJEM03373
AN INTERNATIONAL SCHOLARLY || MULTIDISCIPLINARY || OPEN ACCESS || INDEXING IN ALL MAJOR DATABASE & METADATA

Design and Implementation of Unified Diagnostic Service Protocol

Omkar Shivaji Bhagat
Student at
Department of Electronics and Telecommunication,
SVPM College of Engineering, Baramati, Pune
osbhagat90@gmail.com

Dr. N. A. Doshi
Professor at
Department of Electronics and Telecommunication,
SVPM College of Engineering, Baramati, Pune
nadoshi@engg.svpm.org.in

Abstract- Modern vehicles use the Electronics Control Unit (ECU) to control and monitor all the activities within the vehicle. The number of ECUs are increasing as the complexity of vehicles increases. All the ECUs present in the vehicles are communicated with each other via CAN protocol. Any malfunction in the ECU or abnormal behaviour of ECU is detected or understood by diagnostic services. CAN Protocol does not have advanced features like Diagnostic. The CAN protocol covers only the Physical and Data link layer of the OSI model. There is a need for a standardised diagnostic protocol which can use CAN as underlying technology. Standardised diagnostic protocols used in the automotive domain are On Board Diagnostics (OBD) and Unified Diagnostic services (UDS). UDS protocol is defined under the ISO 14229 standard and provides a standardized framework for in-vehicle communication and fault diagnosis. This project focuses on the design and implementation of the UDS protocol on an embedded system using STM32F407 microcontrollers. The project involves developing a diagnostic communication system between Electronic Control Units (ECUs) Light Control Module (LCM) and Tester tool connected over a CAN bus.

Keywords- Unified Diagnostic Services (UDS),ISO 14229, Electronic Control Unit (ECU),Body Control Module (BCM),Light Control Module (LCM), Controller Area Network (CAN),Diagnostic Trouble Code (DTC),Routine Control Identifier(RID) , Data Identifier (DID),ISO-TP (Transport Protocol)

I. INTRODUCTION (*HEADING 1*)

In modern automotive systems, the ability to diagnose and troubleshoot ECUs efficiently is critical for ensuring vehicle reliability and performance. The UDS protocol, defined by ISO 14229, is widely adopted in the automotive industry to facilitate standardized communication between diagnostic tools and vehicle ECUs over the CAN network. UDS enables functions such as fault detection, software updates, parameter tuning, and remote ECU diagnostics, making it an essential part of vehicle maintenance and repair processes.

This paper presents the design and implementation of a UDS-based diagnostic communication system using STM32F4 microcontrollers. The project involves vehicle ECUs Light Control Module (LCM) that communicate via CAN bus and respond to UDS diagnostic requests. The system supports essential UDS services, including Read Data by Identifier (0x22), Write Data by Identifier (0x2E), Routine Control (0x31) and Read DTC Information (0x19).

To validate the robustness of the system, fault injection techniques are utilized, simulating real-world ECU failures such as Overvoltage and Under-Voltage conditions, ECU power loss, and LED circuit failures. The diagnostic response to these failures is analyzed using a Waveshare USB-to-CAN module. By implementing UDS on embedded automotive ECUs, this project aims to demonstrate the practical application of standardized diagnostic services in real-world vehicle systems, improving the efficiency of vehicle fault detection and maintenance processes.

II. LITERATURE SURVEY

The authors Kataria et al. focus on implementing the UDS protocol over CAN using the ISO 15765-2 (ISOTP) transport layer. The study presents a robust UDS stack supporting services like Request Download and Transfer Data, highlighting ISO-TP's role in handling segmented messages. It emphasizes UDS importance in advanced diagnostics and ECU reprogramming in modern vehicles. [1]

The authors M. Kuntoji, V. Medam and Veena Devi S. V, focus on the design and implementation of the UDS protocol in a fully integrated automotive electronic system. The research explores the role of UDS in managing diagnostics, ECU communication,

and fault code retrieval. The authors present a detailed methodology for developing a UDS stack, with emphasis on key diagnostic services such as Read DTC Information (0x19) and Control DTC Setting (0x85). The study also discusses the challenges faced in resource constrained embedded environments, providing recommendations for optimizing UDS performance in modern automotive ECUs. This paper contributes to the understanding of UDS implementation within realworld automotive systems.[2]

The author S. Desai and Y. Bhateshvar explores the implementation of the UDS protocol on an Arduino platform, using MATLAB to simulate and test the diagnostic services over the CAN bus. The research demonstrates a low-cost approach to understanding UDS by utilizing accessible tools such as Arduino for embedded system design. The authors successfully implemented key UDS services, including Diagnostic Session Control (0x10) and Clear Diagnostic Information (0x14), highlighting the flexibility of UDS for educational and prototyping purposes. This work provides a practical example of UDS implementation in a simplified environment, offering insights into how UDS functions in real time systems.[3]

The paper by Chatterjee et al. (2024) investigates vulnerabilities in diagnostic protocols like UDS used in commercial vehicle networks. It highlights how insecure implementations can be exploited to gain unauthorized ECU access or disrupt vehicle functions. Building on past automotive security research, the authors focus specifically on diagnostic service abuse and demonstrate real-world attack scenarios. Their findings emphasize the need for stronger authentication and access controls in vehicle diagnostics.[4]

The study by Krishnamurthy (2021) provides a comprehensive overview of the UDS protocol used in automotive systems. It explains the structure, key services, and operational flow of UDS in ECU diagnostics and communication. The paper serves as a technical reference for understanding how UDS enables fault detection, ECU programming, and system monitoring. It also emphasizes the importance of proper implementation for reliable and secure diagnostics.[5]

The ISO 142291:2013 document defines the UDS protocol, which standardizes communication between vehicle ECU and external diagnostic tools. UDS is crucial for performing diagnostics, ECU reprogramming, and fault code management in modern vehicles. This standard defines a comprehensive set of services for vehicle diagnostics, such as Diagnostic Session Control (0x10), ECU Reset (0x11), Read Data by Identifier (0x22), and Clear Diagnostic Information (0x14). UDS operates across various communication channels, including CAN, Ethernet, and FlexRay, enabling diagnostic tools to interact with a wide range of vehicle systems. The document ensures that diagnostic tools and ECUs across different manufacturers can communicate using a unified and standardized approach, promoting interoperability in the automotive industry.[6]

The ISO 11898 standard specifies the CAN protocol, which is widely used for communication between ECUs in automotive systems. CAN enables real time, low latency communication in embedded systems, making it ideal for vehicle applications where reliability and data integrity are critical. The standard defines both the CAN protocol for data link and physical layer specifications. CAN is particularly suited for vehicle diagnostics because it allows multiple ECUs to communicate on the same bus, reducing wiring complexity and ensuring prioritized message handling. The combination of UDS over CAN, known as Diagnostic over CAN (DoCAN), allows diagnostic tools to access ECU data, read diagnostic trouble codes (DTCs), and perform other essential diagnostic functions. [7]

The ISO 15765-2 standard, also known as ISO Transport Protocol (ISOTP), provides a communication framework for sending diagnostic messages over the CAN bus. Since CAN frames have a limited data size (typically 8 bytes), larger diagnostic messages such as UDS requests need to be segmented and reassembled. ISOTP ensures the correct transport of UDS messages by defining a method for segmenting large messages, flow control, and message reassembly. This protocol is essential for UDS over CAN, as it enables diagnostic services to send larger payloads efficiently, ensuring communication reliability and data integrity even with the CAN protocol's bandwidth constraints. ISOTP plays a pivotal role in making UDS functional over CAN, as it handles message fragmentation and ensures that services like Request Download (0x34) and Transfer Data (0x36) can be executed seamlessly. [8]

The ISO 13400 standard specifies Diagnostic Communication over Internet Protocol (DoIP), which allows UDS services to be transmitted over Ethernet instead of traditional automotive communication buses like CAN. DoIP offers higher bandwidth, making it ideal for modern vehicles that require faster and more robust diagnostic communication, particularly for large data transfers and remote diagnostics. Ethernet based diagnostic communication provides significant advantages in terms of speed and scalability compared to CAN, making it suitable for more advanced vehicles that support high speed communication, such as electric and autonomous vehicles.[9]

The ISO 26262 standard for Functional Safety in automotive systems emphasizes the importance of reliable diagnostics to ensure that vehicle systems are functioning safely. While this standard is not directly related to UDS or communication protocols like CAN or DoIP, it provides guidelines on ensuring that diagnostic systems, including UDS-based systems, meet stringent safety requirements. Implementing UDS in compliance with ISO 26262 helps ensure that diagnostic communication is reliable, even in safety-critical systems, contributing to the overall functional safety of the vehicle. [10]

III. SYSTEM OVERVIEW.

A. *UDS Protocol Overview*

UDS is a high-level diagnostic communication protocol defined by the ISO 14229 standard. It is extensively used in the automotive industry to facilitate standardized communication between diagnostic tools and vehicle ECU. UDS operates over the CAN, which serves as the underlying low-layer protocol, providing the physical and data link layers necessary for communication.

B. *Client Server Architecture*

UDS protocol follows Client-Server architecture. In Client-Server architecture the diagnostic tool acts as the client, and the ECU acts as the server. The client initiates diagnostic requests. client sends commands to the ECU for actions like reading data, clearing diagnostic trouble codes (DTCs), or performing ECU reprogramming. The client is typically a diagnostic scanner or testing equipment. The server, which is the ECU, processes the client's requests and responds accordingly. It provides data or executes actions as specified by the client's UDS commands, such as responding with vehicle status or resetting the ECU.

C. *UDS Message Structure*

UDS protocol is request Based protocol.

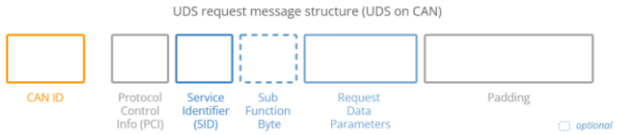


Fig.1. UDS Message Frame Format

- CAN ID: The CAN ID is a unique identifier used in the CAN protocol to distinguish between different messages on the network. In UDS, CAN ID used to identify the source and destination of diagnostic requests and responses. CAN IDs can be either standard (11-bit) or extended (29-bit), depending on the network configuration.
- Protocol Control Information (PCI): The PCI field can be 1-3 bytes long and contains information related to the transmission of messages that do not fit within a single CAN frame. PCI helps with message type identification, segmentation and reassembly, flow control, and sequence numbering.
- Service Identifier (SID): The SID is a unique code that specifies the type of diagnostic service being requested. Each service has a predefined SID, such as 0x22 for Read Data by Identifier or 0x2E for Write Data by Identifier. The SID is crucial for the ECU to understand the nature of the request.
- Sub-Function Byte: The subfunction byte is a single byte value that follows the SID in the UDS message structure. It indicates specific actions or modes within the main service. For example, in the Routine Control service (0x31), the subfunction byte can specify different routines to be executed. In case the response is positive, the tester may want to suppress the response (as it may be irrelevant). This is done by setting the 1st bit to 1 in the sub function byte. Negative responses cannot be suppressed.
- Data Identifier (DID): The DID identifies specific data elements within the ECU that are being accessed or modified. For example, a DID might refer to a particular sensor reading or configuration parameter. The DID ensures that the correct data is targeted by the diagnostic service.
- Data Parameters: Data Parameters are additional data elements required for the service. Parameters can include values to be written, conditions to be met, or specific instructions for the ECU. They provide the necessary context for the SID and DID to perform the requested operation.

D. *UDS Response*

In UDS protocol client trigger request and server will respond to this request as per configuration. The responses are of two types Positive response and Negative response.

1) Positive response

A positive response indicates that the requested diagnostic service has been successfully executed by the ECU. The structure of a positive response typically mirrors the request, including the Service Identifier (SID) with an added offset of 0x40 to distinguish it as a response. For example, if the request SID is 0x22 (Read Data by Identifier), the positive response SID will be 0x62. Positive responses may also include additional data, such as the requested information or confirmation of the action performed.

2) Negative response

A negative response indicates that there was an issue with the requested diagnostic service. Negative responses include a Negative Response Code (NRC) that specifies the type of error encountered. The structure of a negative response includes the original SID, followed by the NRC. Common NRCs include

- 0x10 (General Reject): The request was rejected for a general reason not covered by other NRCs.
- 0x11 (Service Not Supported): The requested service is not supported by the ECU.
- 0x12 (Sub Function Not Supported): The specified sub function is not supported by the ECU.
- 0x13 (Incorrect Message Length or Invalid Format): The message length or format is incorrect.

E. *Block Diagram*

The block diagram illustrates the architecture of the diagnostic communication system implemented using the UDS protocol on STM32 microcontrollers. The system consists of several key components connected in sequence to facilitate efficient communication and control.

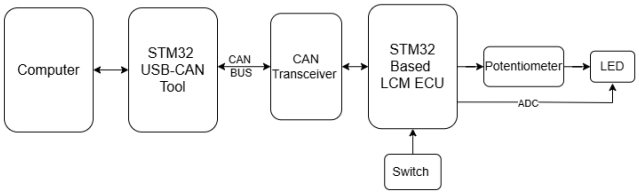


Fig. 2. UDS implementation Block Diagram

1) Hardware Components

- Computer: The computer acts as the diagnostic tool. Waveshare USB-CAN-A software tool is installed on the computer. Computer is interfacing with the STM32 USB-CAN tool. It sends diagnostic requests and receives responses, enabling the monitoring and control of the ECUs.
- Waveshare USB-CAN-A Tool: This tool serves as the interface between the computer and the CAN network. It converts USB signals from the computer into CAN signals and vice versa, allowing seamless communication between the diagnostic tool and the ECUs. The Waveshare USB-CAN-A supports various CAN baud rates and provides a reliable connection for diagnostic operations.
- CAN Transceiver: The CAN transceiver is responsible for transmitting and receiving CAN signals between the Waveshare USB-CAN-A tool and the ECUs. It ensures reliable data transmission over the CAN bus.
- STM32 Based LCM ECU: The Light Control Module (LCM) ECU is based on the STM32 microcontroller. It processes diagnostic requests related to lighting control and responds accordingly. The LCM ECU is connected to input and output devices to perform its functions.
- Potentiometer: The potentiometer is connected between the LED and an output GPIO pin of the STM32 Based LCM ECU. It is used to simulate conditions such as under-voltage and overvoltage to create faulty conditions
- LED: The LED is connected to the GPIO Pin of ECU via Potentiometer. It provides visual feedback based on the potentiometer's settings, allowing for real-time monitoring of light control and fault
- Switch: The switch is connected to the STM32 Based LCM ECU and serves as an input device. It is used to turn on and off LEDs.

2) Software Components

- STM32CubeIDE: STM32CubeIDE is an integrated development environment (IDE) provided by STMicroelectronics. It is an Eclipse-based development environment, offering a comprehensive platform for developing, debugging, and testing applications on STM32 microcontrollers. STM32CubeIDE supports code generation, project management, and debugging features.
- STM32CubeMX: STM32CubeMX is a graphical tool that simplifies the configuration and initialization of STM32 microcontrollers. It allows developers to configure peripherals, middleware, and pin assignments through an intuitive interface. STM32CubeMX generates initialization code that can be directly used in STM32CubeIDE, streamlining the development process.
- Waveshare USB-CAN-A: It is a software tool by Waveshare. This tool is used to send and receive the CAN message on the computer. This tool can configure baud rate of CAN protocol. This is the interface for UDS request and Response.

IV. FLOW CHART

A. *Read Data By Identifier (0x22)*

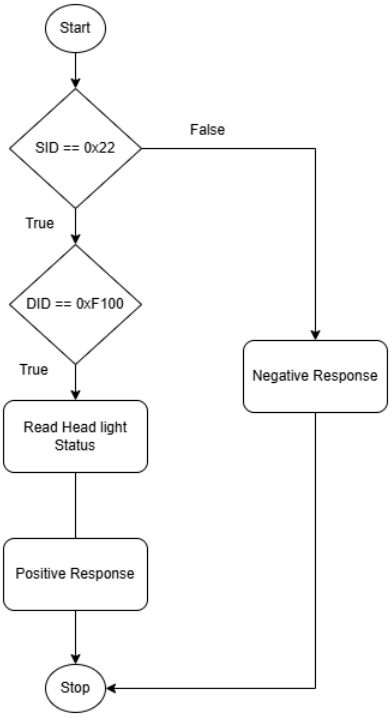


Fig.3. Flowchart for Read Data by Identifier

In the flow chart for UDS Service 0x22 (Read Data by Identifier), the process begins with the ECU receiving a request message that includes Service ID 0x22 and a DID (Data Identifier) to read. The ECU then checks if the requested DID is supported. If the DID is valid, the ECU retrieves the corresponding data from memory or storage associated with that DID and sends a positive response with Service ID 0x62 followed by

the DID and its value. If the DID is unsupported or unavailable, the ECU responds with a negative response (0x7F) and an appropriate error code. The flow then returns to an idle or wait state until a new request is received.

B. Write Data By Identifier

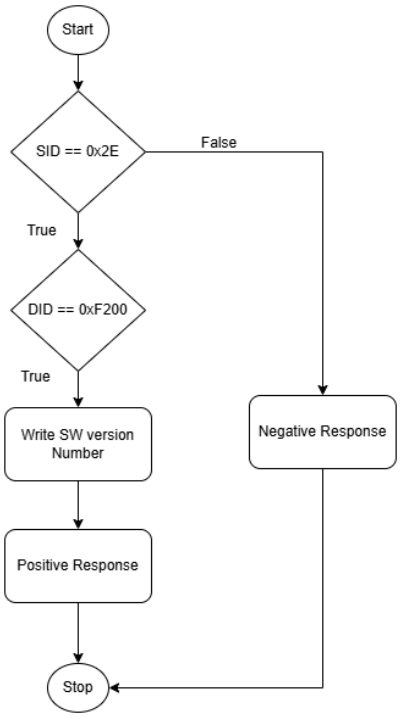


Fig.4. Flowchart for Read Data by Identifier

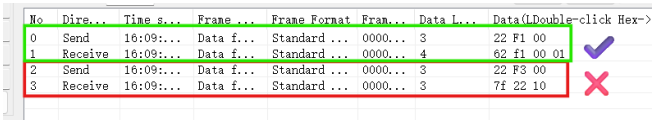
The flow chart for the UDS Service 0x2E (Write Data by Identifier) begins with the ECU receiving a request containing the DID (Data Identifier) and data to write. The ECU first validates the DID to check if it's supported. If the DID is valid, it proceeds to write the data to the specified memory location or register. After writing, the ECU sends a positive response with a response code (0x6E or 0x62), indicating success. If the DID is invalid or the data cannot be written, the ECU sends a negative response (0x7F) with an error code, such as "Request Out of Range" (0x31) or "Sub-function Not Supported" (0x12), before ending the process. This flow ensures data integrity by verifying the DID and handling errors gracefully.

V. RESULT

The project aims to implement a fully functional UDS protocol over the CAN bus on an STM32 microcontroller. The expected results include not only the correct functioning of the UDS protocol stack but also the successful execution of UDS services with proper request and response handling. Below are the key services that will be implemented, along with their corresponding request and response formats.

1) Read Status of LED

The current status of LED can be retrieved using service read Data by Identifier (0x22) and DID (0xF100)

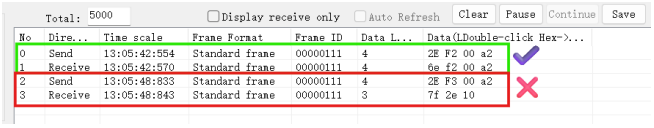


No	Dire...	Time s...	Frame ...	Frame Format	Fram...	Data L...	Data(LDouble-click Hex->
0	Send	16:09:...	Data f...	Standard ...	0000...	3	22 F1 00
1	Receive	16:09:...	Data f...	Standard ...	0000...	4	62 f1 00 01
2	Send	16:09:...	Data f...	Standard ...	0000...	3	22 F3 00
3	Receive	16:09:...	Data f...	Standard ...	0000...	3	7f 22 10

Fig. 5. Request and Response for service (0x22)

2) Write Software Version Number

The updated SW version number can be written into the Software using Write Data by identifier (0x2E) and DID (0xF200)



Total: 5000 Display receive only Auto Refresh Clear Pause Continue Save

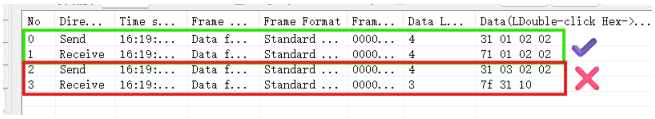
No	Dire...	Time scale	Frame Format	Frame ID	Data L...	Data(LDouble-click Hex->...
0	Send	13:05:42:554	Standard frame	00000111	4	2E F2 00 a2
1	Receive	13:05:42:570	Standard frame	00000111	4	6e f2 00 a2
2	Send	13:05:48:833	Standard frame	00000111	4	2E F3 00 a2
3	Receive	13:05:48:843	Standard frame	00000111	3	7f 2e 10

Fig.6. Request and Response for service (0x2E)

3) Blinking LED for Testing

To check whether the LED is working correctly or not, a dedicated blinking led routine can be used with Routine control SID (0x31).

To start routine sub–Function SID (0x01) and to stop routine sub function SID (0x02) is used. For Blinking led with 1 sec delay routine is used 0x2020.

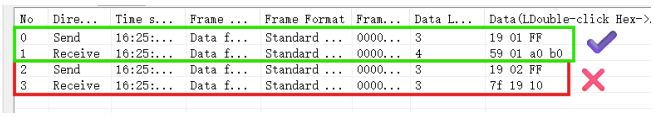


No	Dire...	Time s...	Frame ...	Frame Format	Fram...	Data L...	Data(LDouble-click Hex->...
0	Send	16:19:...	Data f...	Standard ...	0000...	4	31 01 02 02
1	Receive	16:19:...	Data f...	Standard ...	0000...	4	71 01 02 02
2	Send	16:19:...	Data f...	Standard ...	0000...	4	31 03 02 02
3	Receive	16:19:...	Data f...	Standard ...	0000...	3	7f 31 10

Fig.9. Request and Response for service (0x31)

4) Read DTC to Detect Fault

The fault occurred because voltage and under voltage are stored in a system with a unique number called Diagnostic Trouble Code (DTC). The fault in the system can be understood by reading the stored DTCs using Read DTC service (0x19) and Sub Functional SID DTC by Status Mask (0x01) and status mask value (0xFF). This request will read all the DTC regardless of Status.



No	Dire...	Time s...	Frame ...	Frame Format	Fram...	Data L...	Data(LDouble-click Hex->...
0	Send	16:25:...	Data f...	Standard ...	0000...	3	19 01 FF
1	Receive	16:25:...	Data f...	Standard ...	0000...	4	59 01 a0 b0
2	Send	16:25:...	Data f...	Standard ...	0000...	3	19 02 FF
3	Receive	16:25:...	Data f...	Standard ...	0000...	3	7f 19 10

Fig.8. Request and Response for service (0x19)

VI. CONCLUSION

In conclusion, this Design and Implementation of UDS Protocol for Automotive Diagnostic project successfully implemented diagnostic communication on an STM32F407 controller to interface with a Body Control Module (BCM) and a Lighting Control Module (LCM) over CAN Bus. Key UDS services, including Read Data by Identifier (0x22), Write Data by Identifier (0x2E), and Routine Control (0x31), were developed to manage and monitor BCM and LCM functionalities,

such as reading status data, configuring parameters, and controlling routines like turning headlights on or off. The CAN Bus enabled reliable communication between ECUs, simulating realistic automotive diagnostic scenarios. Overall, this project provides a foundational UDS setup for testing, monitoring, and troubleshooting BCM and LCM systems within a CAN network, adhering to industry diagnostic standards.

REFERENCES

[1] [1] U. Kataria, K. Panchal, J. Puvvada, S. Kadlag and S. Shinde, " Implement Standard UDS Diagnostics Over Can for Automotive Actuator ECU", 2024 International Journal for Multidisciplinary Research (IJFMR), Mumbai, India, E-ISSN: 2582-2160, IJFMR240217526.

[2] [2] M. Kuntoji, V. Medam and Veena Devi S. V, " Design of UDS Protocol in an Automotive Electronic Control Unit", 2023 Recent Developments in Electronics and Communication Systems, Bangalore, India, doi:10.3233/ATDE221266

[3] [3] S. Desai and Y. Bateshwar, "Development of unified diagnostic services on CAN using MATLAB and Arduino", 2023 Materials Today: Proceedings 72 (2023) 1935–1942, Pune, India, https://doi.org/10.1016/j.matpr.2022.10.157

[4] [4] R. Chatterjee, C. Green and J. Daily, "Exploiting Diagnostic Protocol Vulnerabilities on Embedded Networks in Commercial Vehicles", 2024, Symposium on Vehicles Security and Privacy (VehicleSec) 2024, USA, ISBN 979-8-9894372-7-6.

[5] [5] V. N. D. Krishnamurthy, " Unified Diagnostic Services (UDS) in Automotive: A technical Study", 2021, European Journal of Advances in Engineering and Technology, 2021, 8(6):70-73, USA, ISSN: 2394-658X

[6] [6] ISO-14229: International Standard - Road vehicles - Unified diagnostic services (UDS), 2006.

[7] [7] ISO 11898-1:2024 - Road vehicles – Controller area network (CAN) — Part 1: Data link layer and physical coding sublayer.

[8] [8] ISO 15765-2:2016 Road vehicles – Diagnostic communication over Controller Area Network (DoCAN) -- Part 2: Transport protocol and network layer services.

[9] [9] ISO 13400-1:2011-Road vehicles – Diagnostic communication over Internet Protocol (DoIP): — Part 1: General information and use case definition. — Part 2: Transport protocol and network layer services.

[10] [10] ISO 26262-1:2011- Road vehicles – Functional Safety