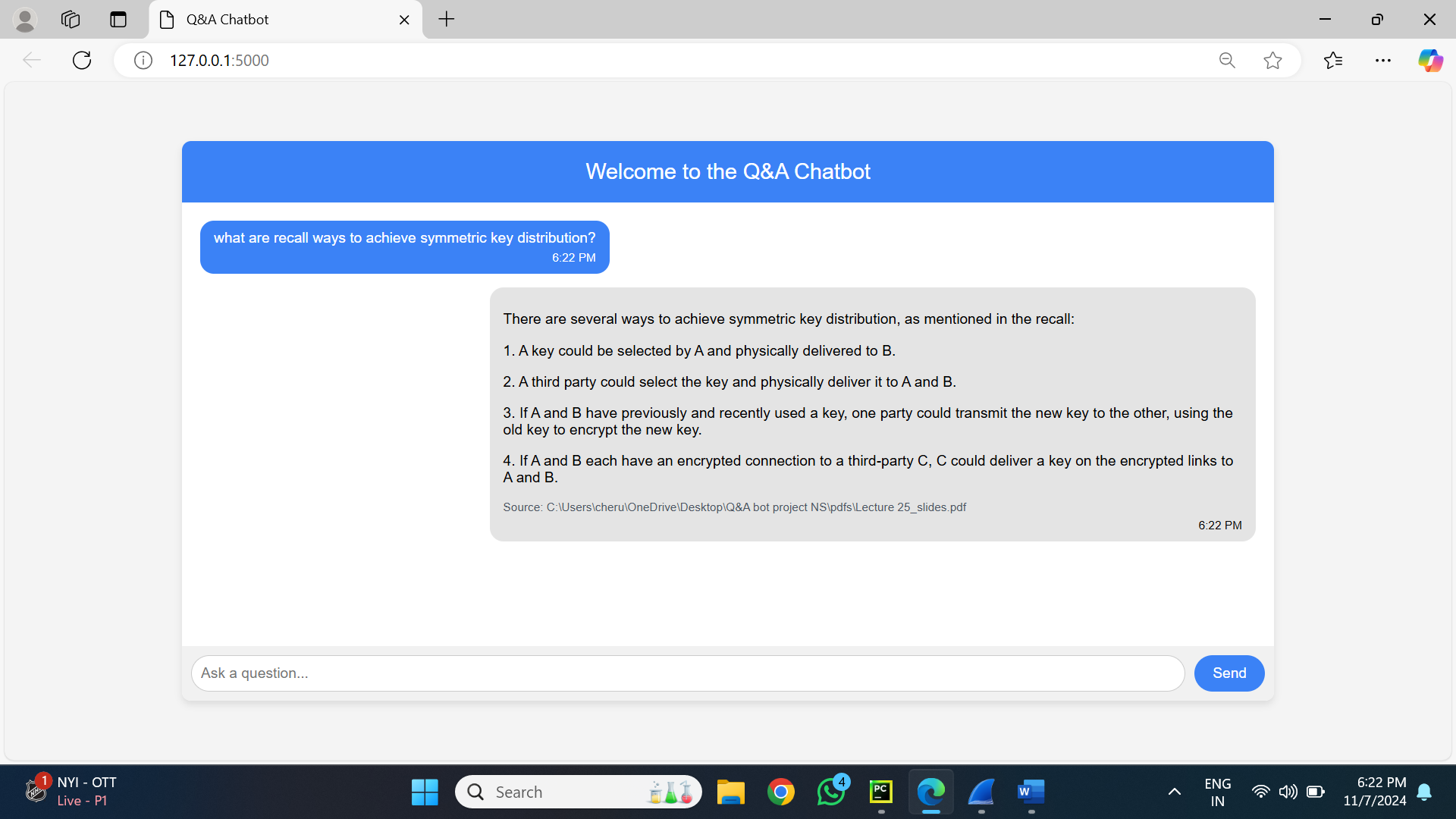
**CS5342 Network Security**

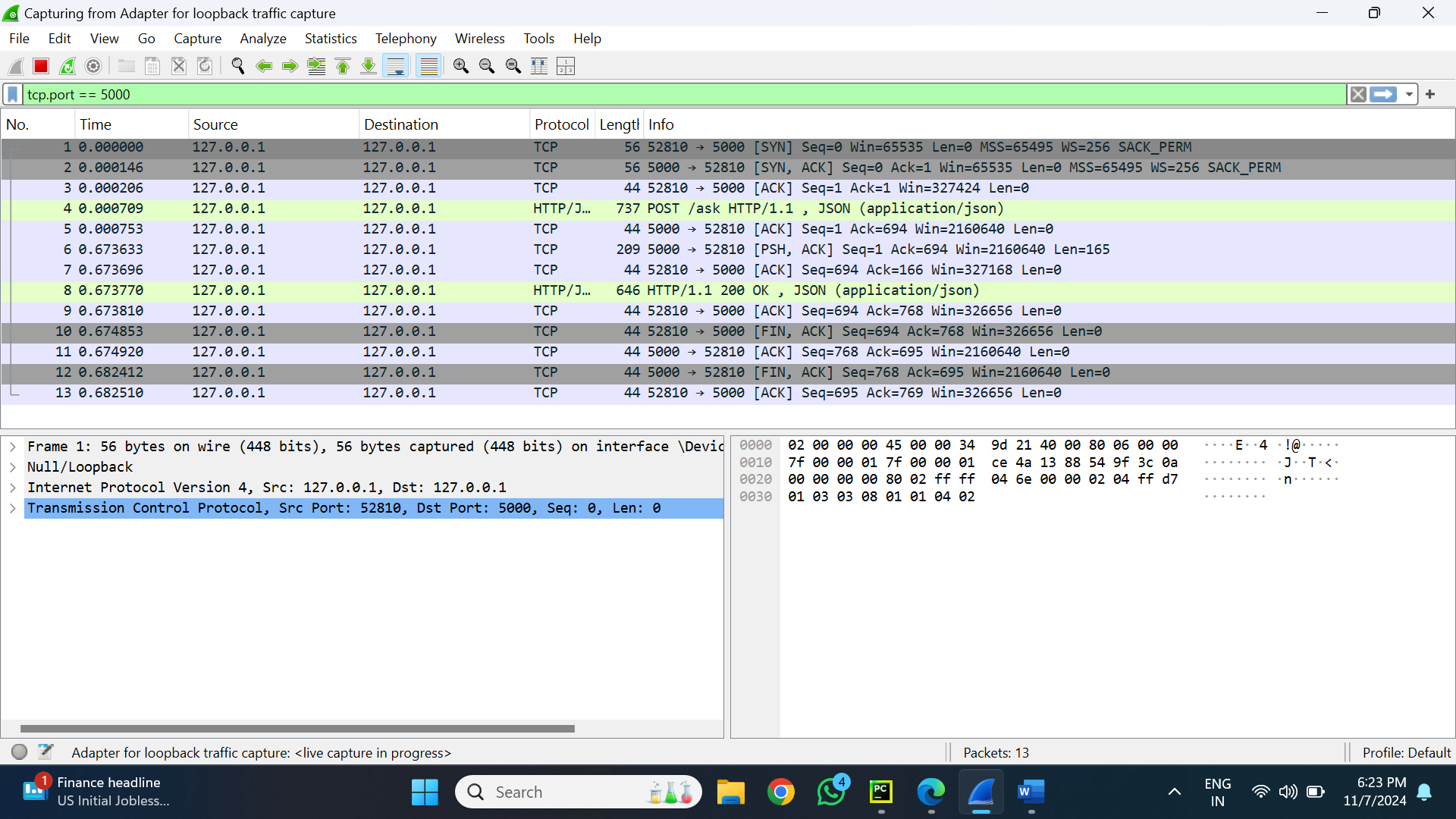
**OnDemand Professor Q&A Bot Project**

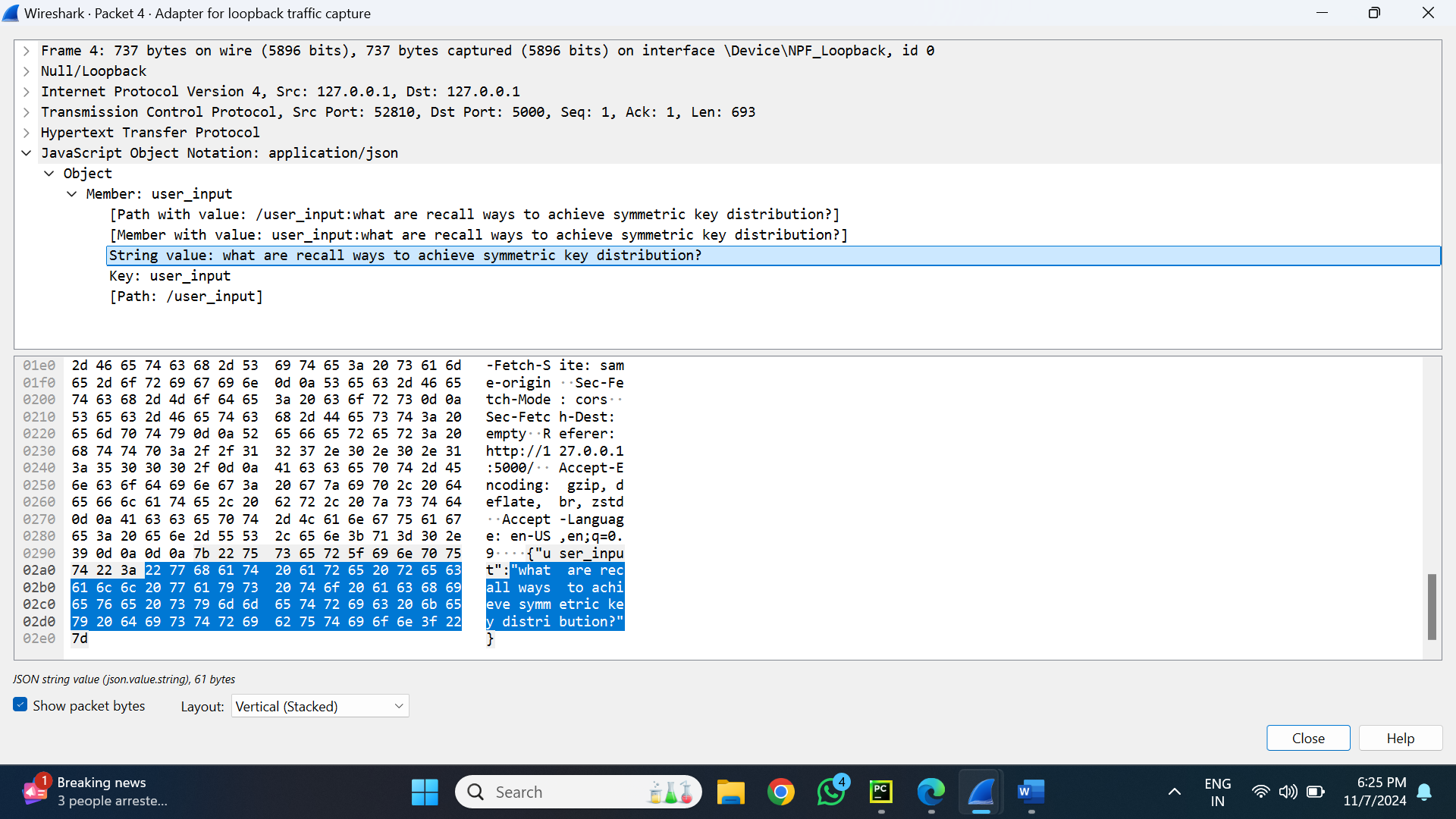
**Name: Greeshma Kaligithi**

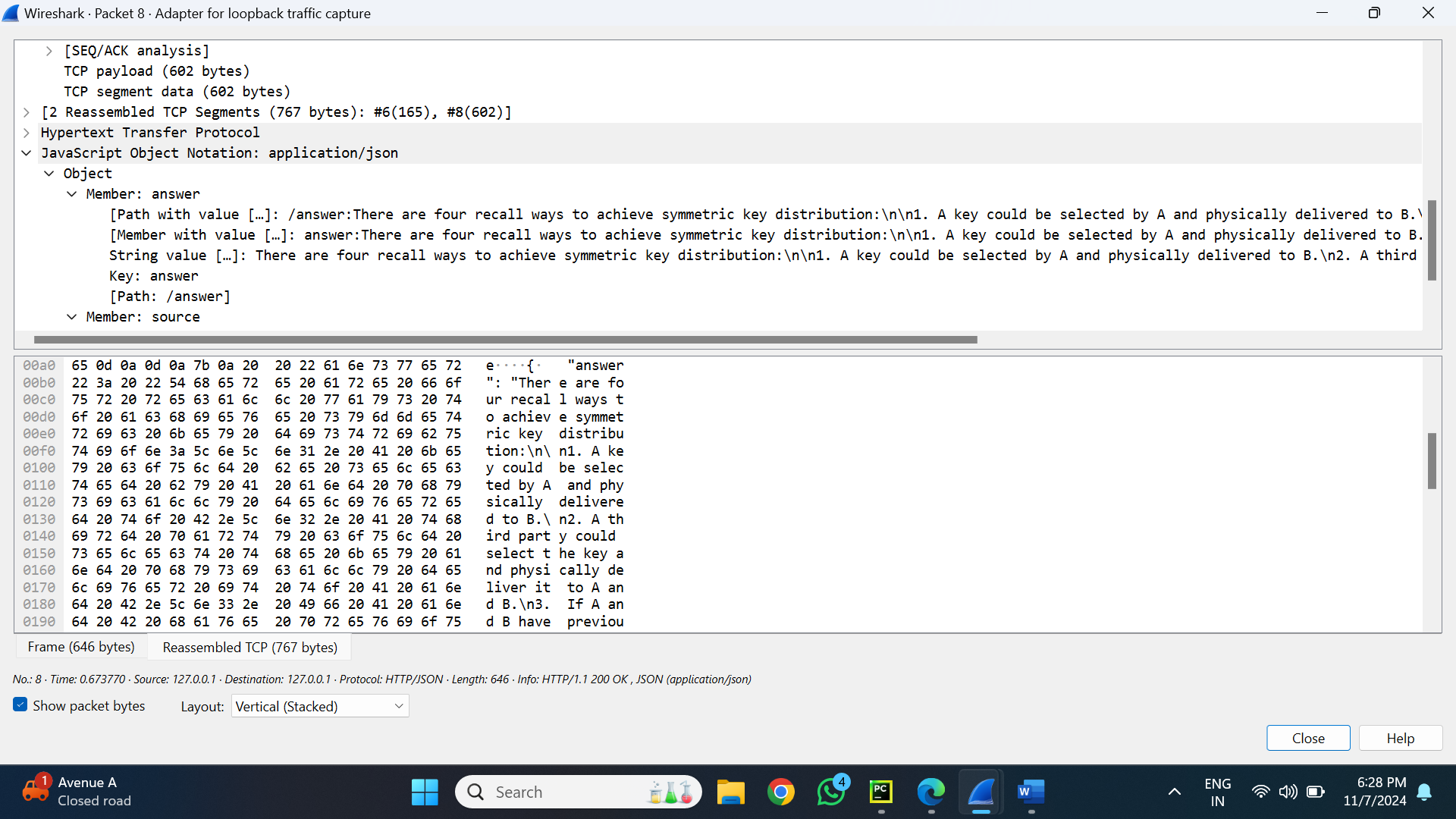
**R ID: R11893902**

* **Prompt 1:**









**Overview of the Prompt 1**

For this prompt a chatbot interface's interactions with a backend LLM server for a particular prompt must be recorded and analysed. This article describes how the user's prompt (Step 1) and the network data recorded during the process (Step 4) are mapped. Wireshark was used to capture the data flow.

**Step 1: User's Prompt**

* The user starts the query via the chatbot interface with the prompt:  
  **"What are recall ways to achieve symmetric key distribution?"**
* This input is the starting point, which gets transmitted to the LLM server via an API request.

**Step 2: Request Data Capture (Wireshark: Packet 4)**

* **Network Details:**
  + **Source**: 127.0.0.1 (localhost)
  + **Destination**: 127.0.0.1 (localhost)
  + **Source Port**: 52810
  + **Destination Port**: 5000
* **Request Payload:** The payload transmitted contains the following JSON structure:

{

"user\_input": "What are recall ways to achieve symmetric key distribution?"

}

• This data shows how the input entered by the chatbot is converted into a structured request that is submitted to the backend server.

K**ey Observations in Request:**

* The **user\_input** field is directly mapped to the user's original query.
* The data travels via a TCP connection within the same machine.

**Step 3: Response Data Capture (Wireshark: Packet 8)**

* **Network Details:**
  + **Source**: 127.0.0.1 (localhost)
  + **Destination**: 127.0.0.1 (localhost)
  + **Source Port**: 5000
  + **Destination Port**: 52810
* **Response Payload:** The response JSON received from the backend server is:

{

"answer": "There are four recall ways to achieve symmetric key distribution:\n1. A key could be selected by A and physically delivered to B.\n2. A third party could select the key and physically deliver it to A and B.\n3. If A and B have previously used a key, one party could transmit the new key to the other using the old key to encrypt the new key.\n4. If A and B each have an encrypted connection to a third party C, C could deliver a key on the encrypted links to A and B."

}

* **Key Observations in Response:**

1. A thorough explanation that corresponds directly to the user's query is provided in the answer field.

2) The response offers numbered points along with an organized and educational response.

**Step 4: Mapping Between Prompt and Trace Data**

* **Prompt (User Input):**

The user's original query: **"What are recall ways to achieve symmetric key distribution?"**

**Mapped Request (Packet 4):**

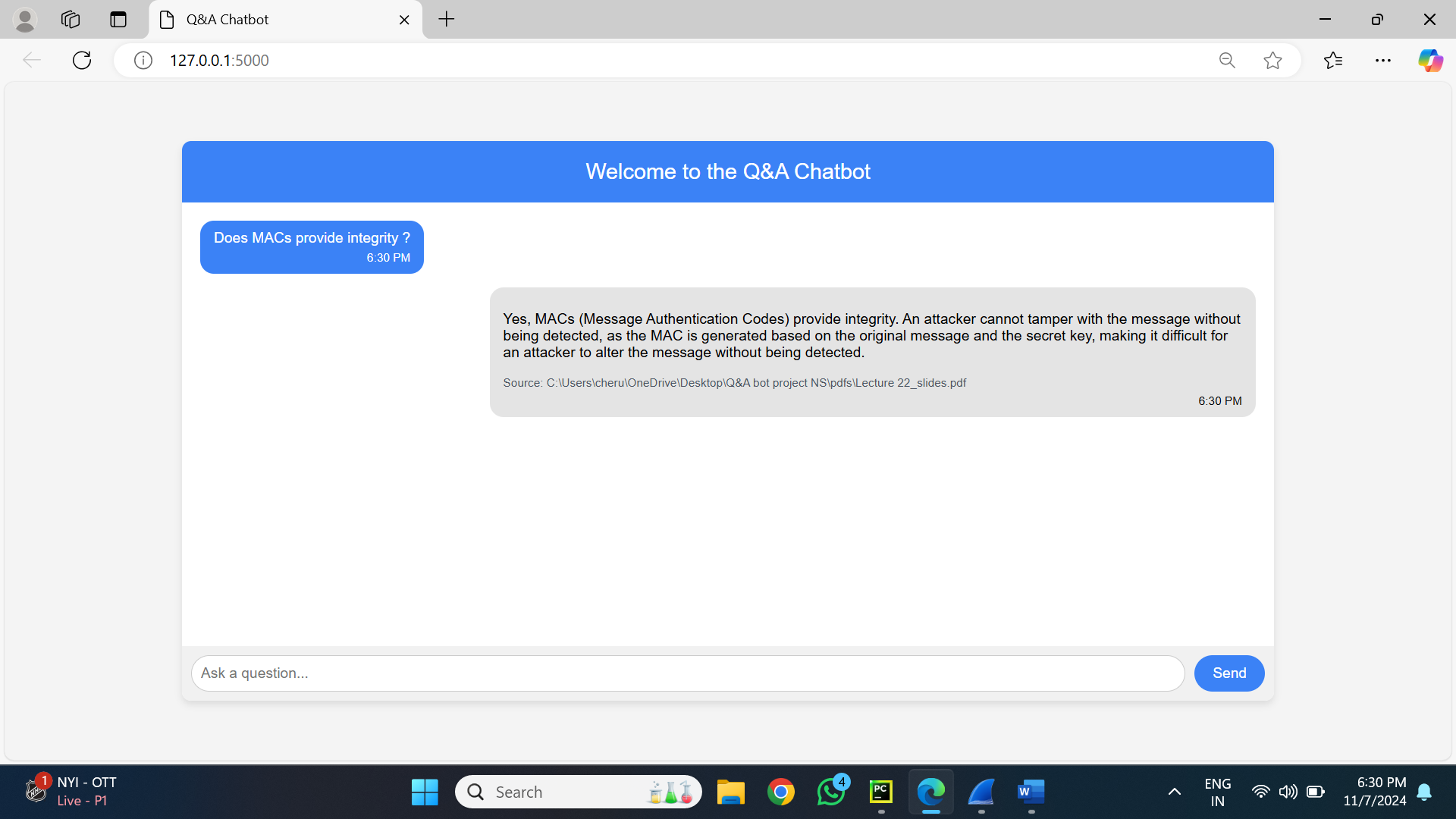
* The **user\_input** field in the JSON payload contains the exact text of the prompt.

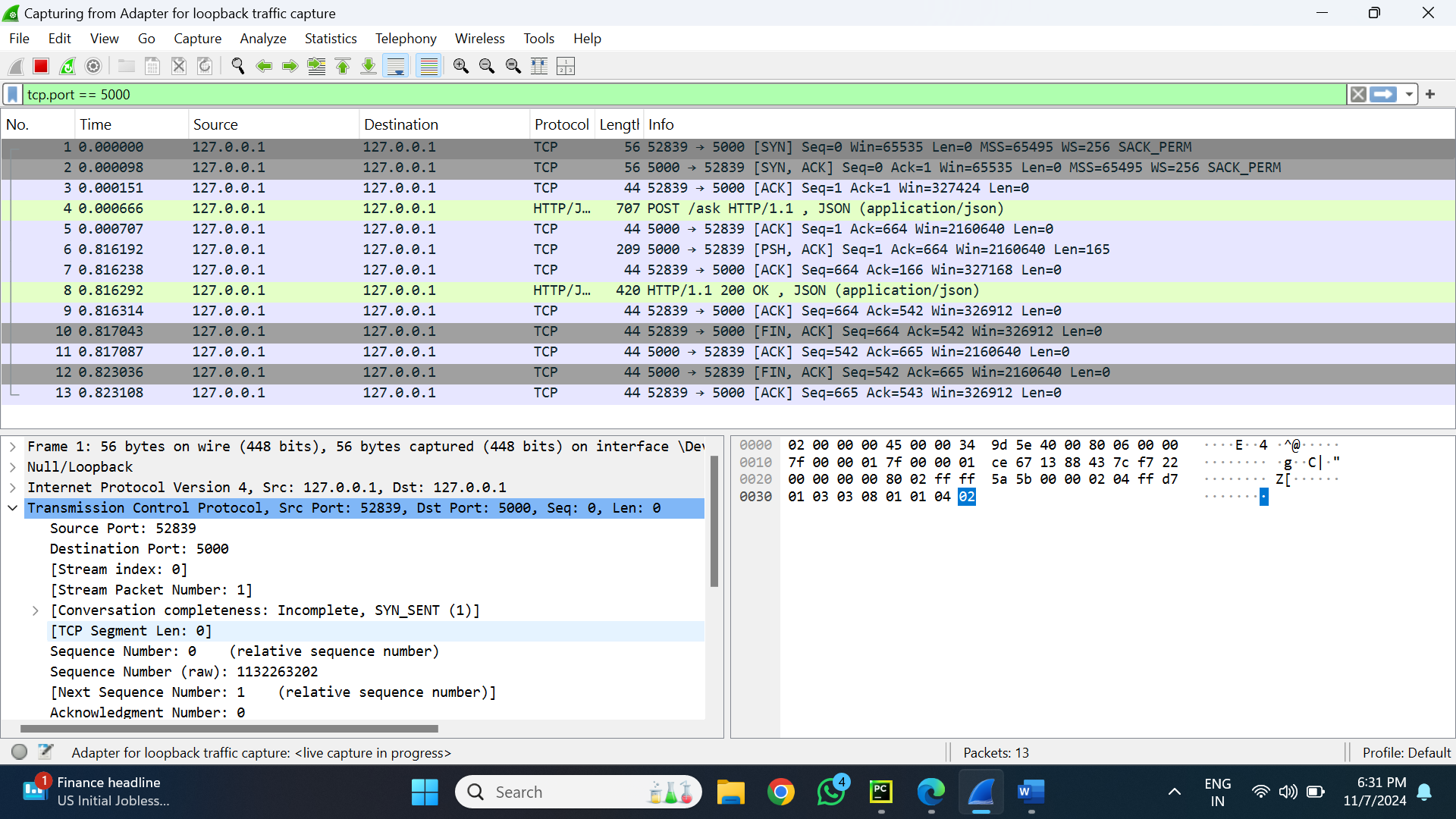
**Mapped Response (Packet 8):**

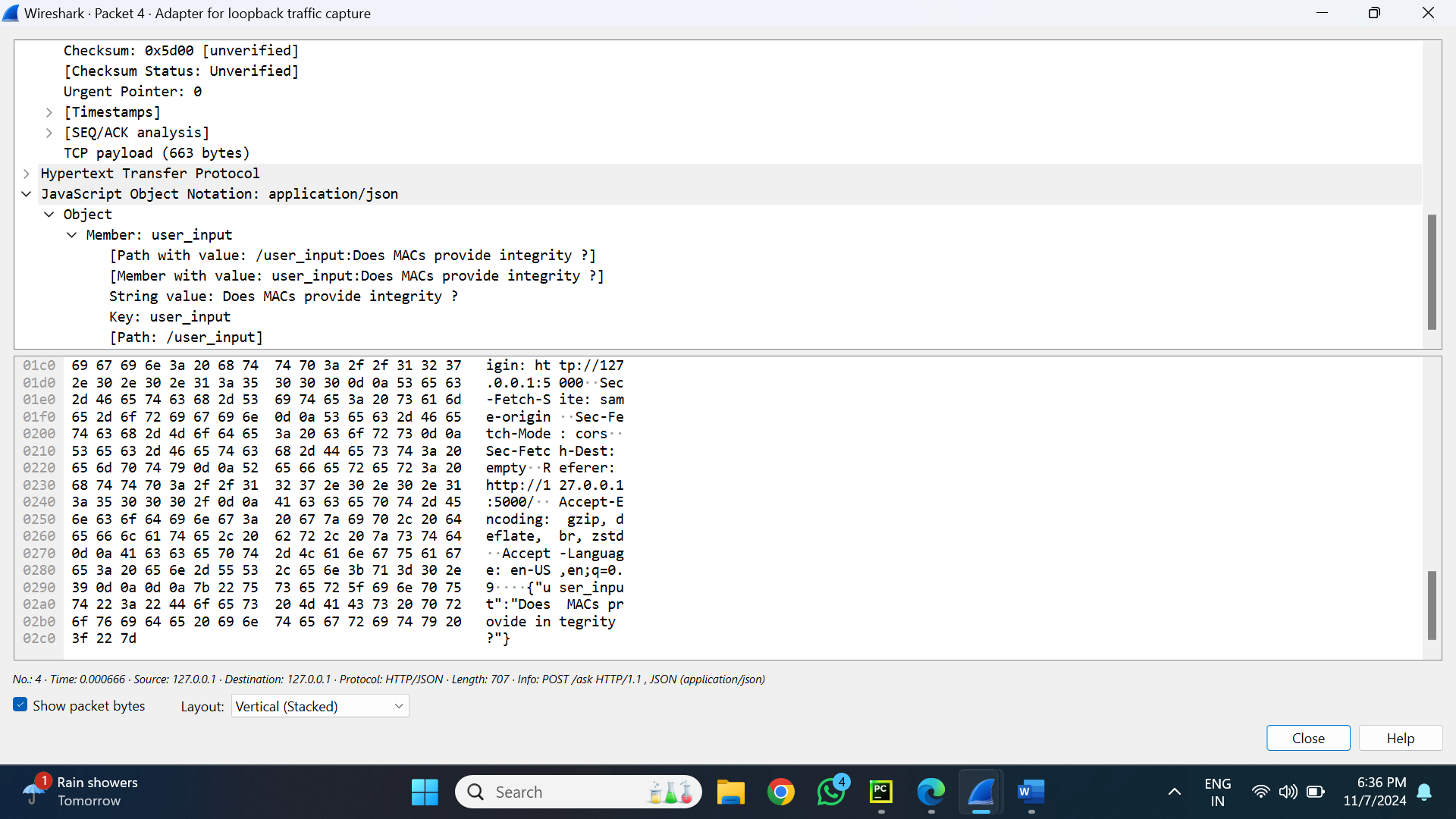
* The **answer** field in the JSON payload contains the server's response, which addresses the query in a structured manner.

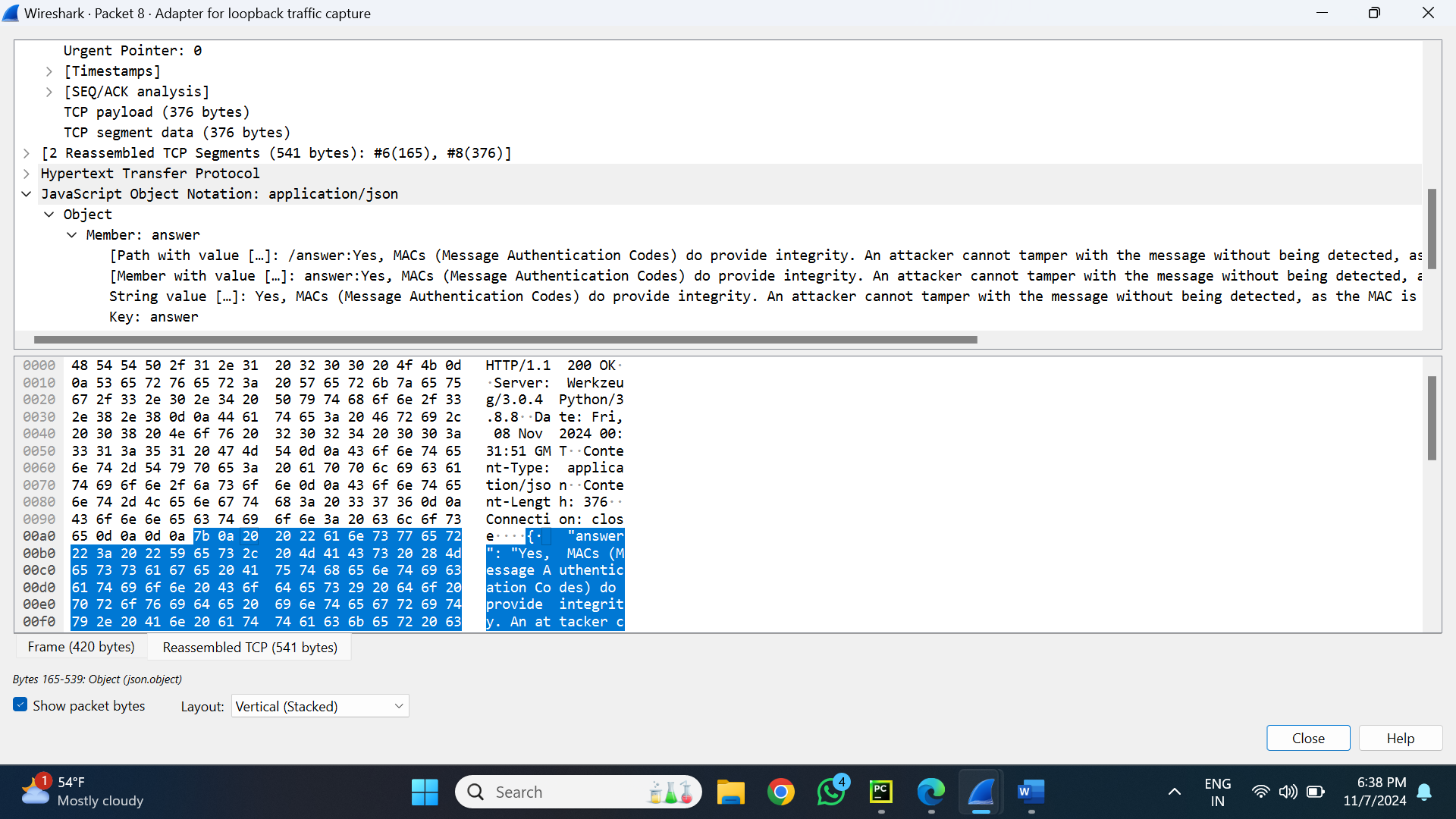
**Summary of Mapping:** The prompt has been summarized in the **user\_input** field of the request payload, processed by the backend server, and returned as a response in the **answer** field.

* **Prompt 2**









**Overview of the Prompt 2**

For this prompt a chatbot interface's interactions with a backend LLM server for a particular prompt must be recorded and analysed. This article describes how the user's prompt (Step 1) and the network data recorded during the process (Step 4) are mapped. Wireshark was used to capture the data flow.

**Step 1: User's Prompt**

The user initiated the query via the chatbot interface with the prompt: **"Does MACs provide integrity?"**  
This input is the starting point, which gets transmitted to the LLM server via an API request.

**Step 2: Request Data Capture (Wireshark: Packet 4)**

**Network Details:**

* **Source:** 127.0.0.1 (localhost)
* **Destination:** 127.0.0.1 (localhost)
* **Source Port:** 52839
* **Destination Port:** 5000

**Request Payload:**

The payload transmitted contains the following JSON structure:

{

"user\_input": "Does MACs provide integrity?" }

**Key Observations in Request:**

* The **user\_input** field is directly mapped to the user's original query.
* The data travels via a TCP connection within the same machine.

**Step 3: Response Data Capture (Wireshark: Packet 8)**

**Network Details:**

* **Source:** 127.0.0.1 (localhost)
* **Destination:** 127.0.0.1 (localhost)
* **Source Port:** 5000
* **Destination Port:** 52839

**Response Payload:** The response JSON received from the backend server is:

{

"answer": "Yes, MACs (Message Authentication Codes) do provide integrity. An attacker cannot tamper with the message without being detected, as the MAC is generated based on the original message and the secret key, making it difficult for an attacker to alter the message without being detected."

}

**Key Observations in Response:**

* The **answer** field contains a detailed explanation, mapping directly to the user's query.
* The response provides a structured and informative answer, explaining how MACs (Message Authentication Codes) ensure message integrity.

**Step 4: Mapping Between Prompt and Trace Data**

**Prompt (User Input):** The user's original query:  
**"Does MACs provide integrity?"**

**Mapped Request (Packet 4):**

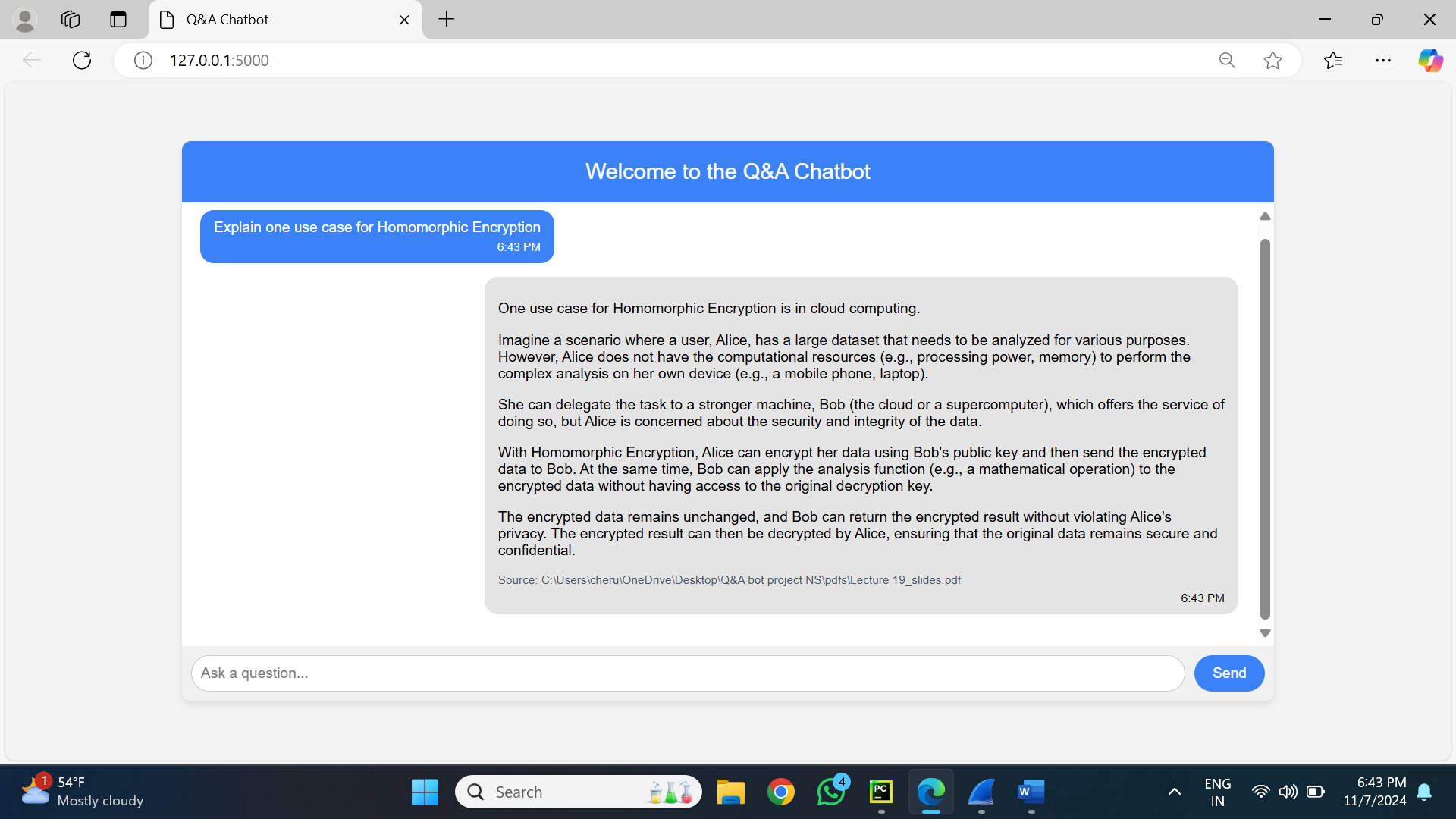
The **user\_input** field in the JSON payload contains the exact text of the prompt.

**Mapped Response (Packet 8):**

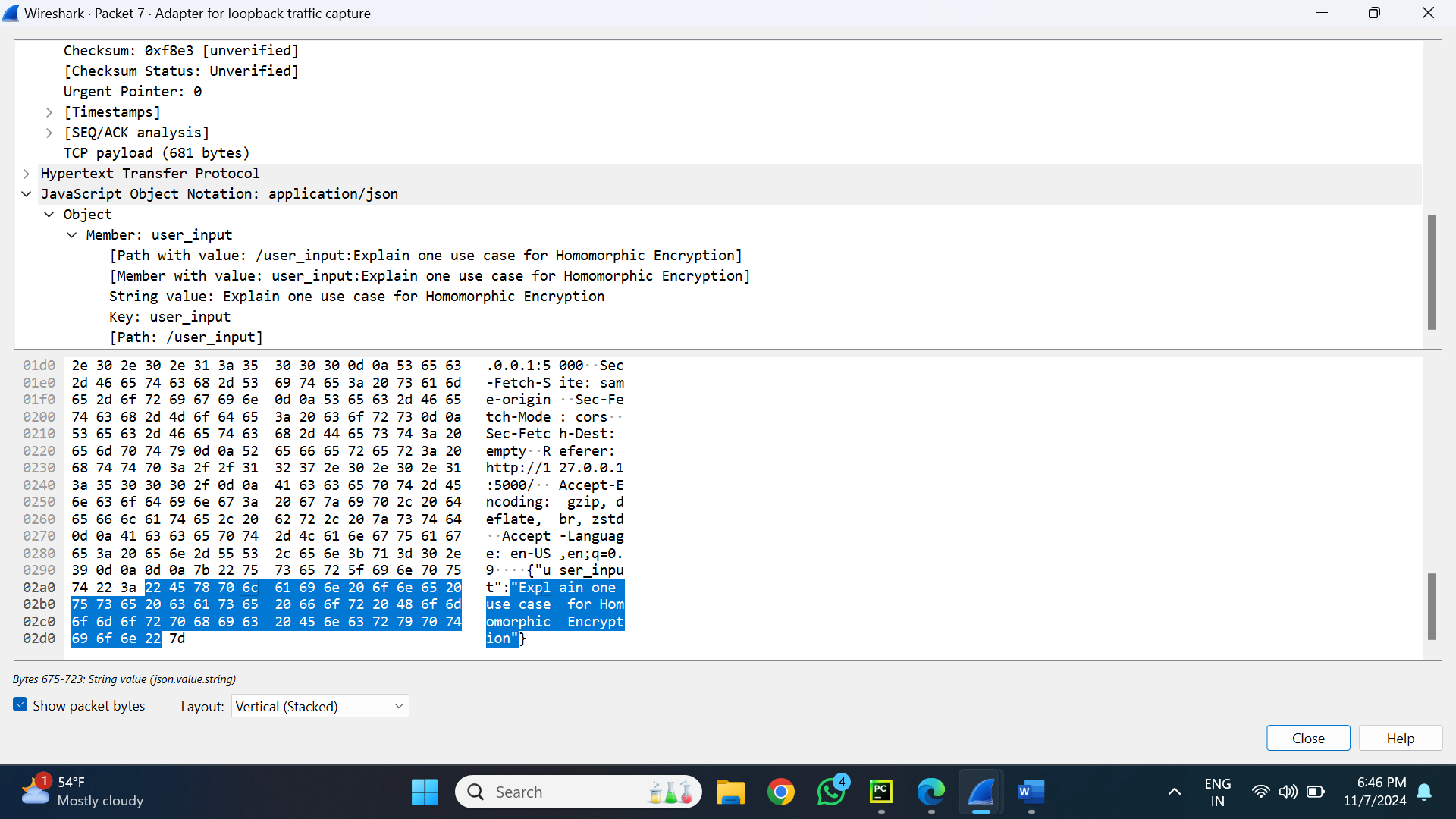
The **answer** field in the JSON payload contains the server's response, which addresses the query in a structured manner.

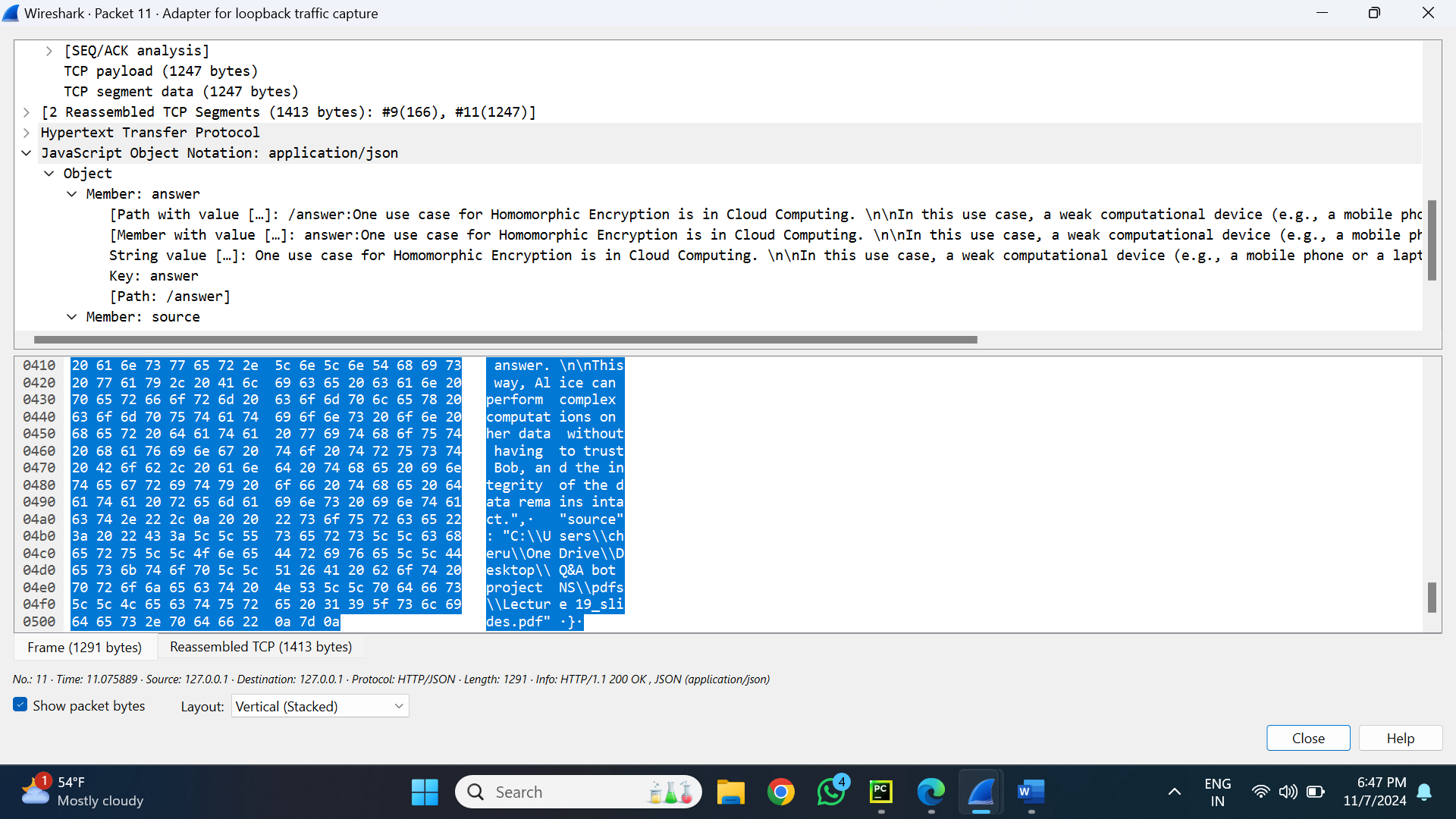
**Summary of Mapping:** The prompt has been summarized in the **user\_input** field of the request payload, processed by the backend server, and returned as a response in the **answer** field.

* **Prompt 3:**









**Overview of the Prompt 3**

For this prompt a chatbot interface's interactions with a backend LLM server for a particular prompt must be recorded and analysed. This article describes how the user's prompt (Step 1) and the network data recorded during the process (Step 4) are mapped. Wireshark was used to capture the data flow.

**Step 1: User's Prompt** The user initiated the query via the chatbot interface with the prompt:  
**"Explain one use case for Homomorphic Encryption."**  
This input is the starting point, which gets transmitted to the LLM server via an API request.

**Step 2: Request Data Capture (Wireshark: Packet 7)**

**Network Details:**

* **Source:** 127.0.0.1 (localhost)
* **Destination:** 127.0.0.1 (localhost)
* **Source Port:** 52868
* **Destination Port:** 5000

**Request Payload:** The payload transmitted contains the following JSON structure:

{

"user\_input": "Explain one use case for Homomorphic Encryption"

}

**Key Observations in Request:**

* The **user\_input** field is directly mapped to the user's original query.
* The data travels via a TCP connection within the same machine.

**Step 3: Response Data Capture (Wireshark: Packet 11)**

**Network Details:**

* **Source:** 127.0.0.1 (localhost)
* **Destination:** 127.0.0.1 (localhost)
* **Source Port:** 5000
* **Destination Port:** 52868

**Response Payload:** The response JSON received from the backend server is:

{

"answer": "One use case for Homomorphic Encryption is in Cloud Computing. Imagine a scenario where a user, Alice, has a large dataset that needs to be analyzed for various purposes. However, Alice does not have the computational resources (e.g., processing power, memory) to perform the complex analysis on her own device (e.g., a mobile phone, laptop). She can delegate the task to a stronger machine, Bob (the cloud or a supercomputer), which offers the service of doing so, but Alice is concerned about the security and integrity of the data. With Homomorphic Encryption, Alice can encrypt her data using Bob's public key and then send the encrypted data to Bob. At the same time, Bob can apply the analysis function (e.g., a mathematical operation) to the encrypted data without having access to the original decryption key."

}

**Key Observations in Response:**

* The **answer** field contains a detailed explanation, mapping directly to the user's query.
* The response provides a real-world example of how Homomorphic Encryption is useful for preserving data privacy in cloud computing.

**Step 4: Mapping Between Prompt and Trace Data**

**Prompt(UserInput):** The user's original query:  
**"Explain one use case for Homomorphic Encryption."**

**Mapped Request (Packet 7):**

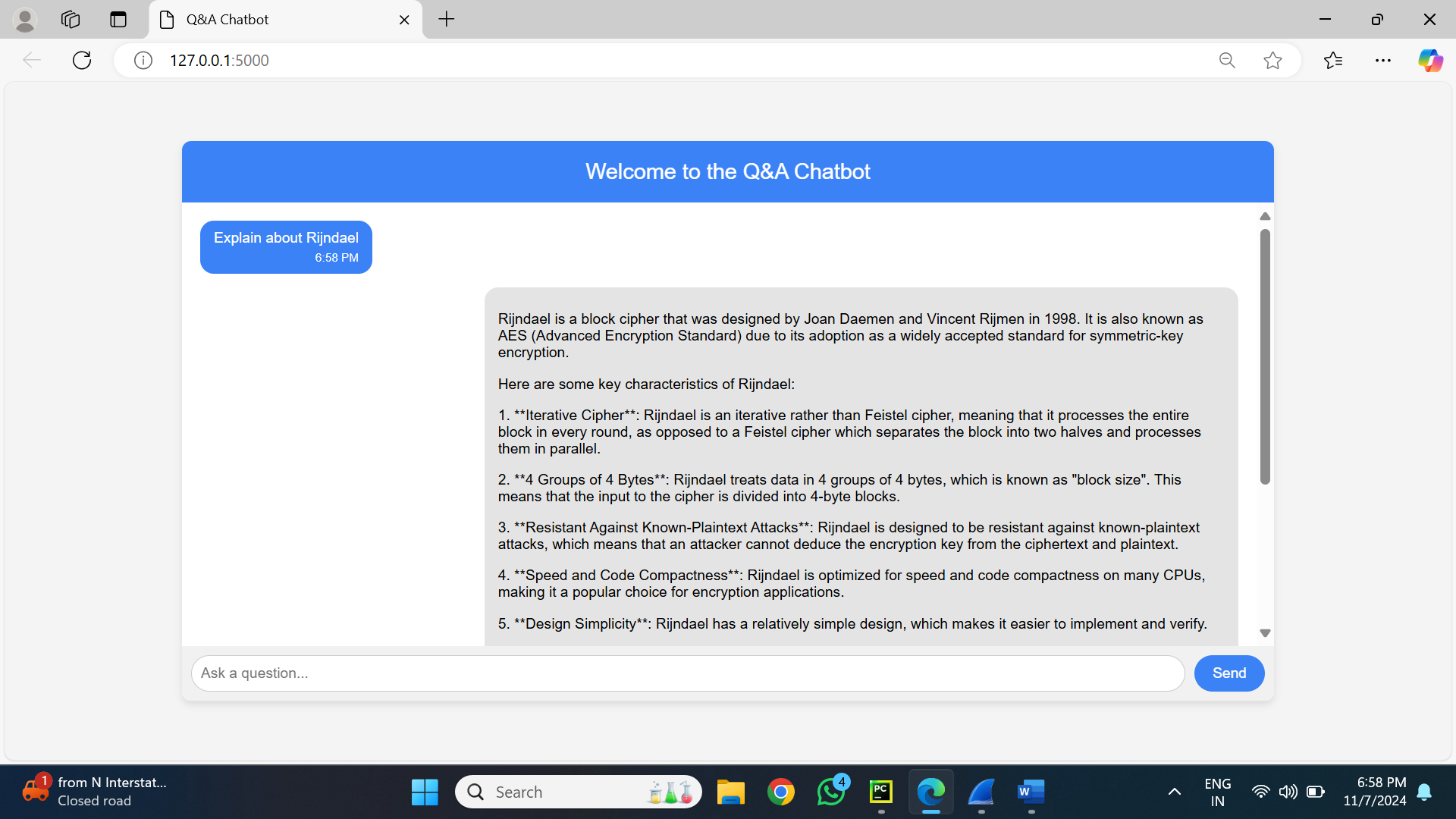
The **user\_input** field in the JSON payload contains the exact text of the prompt.

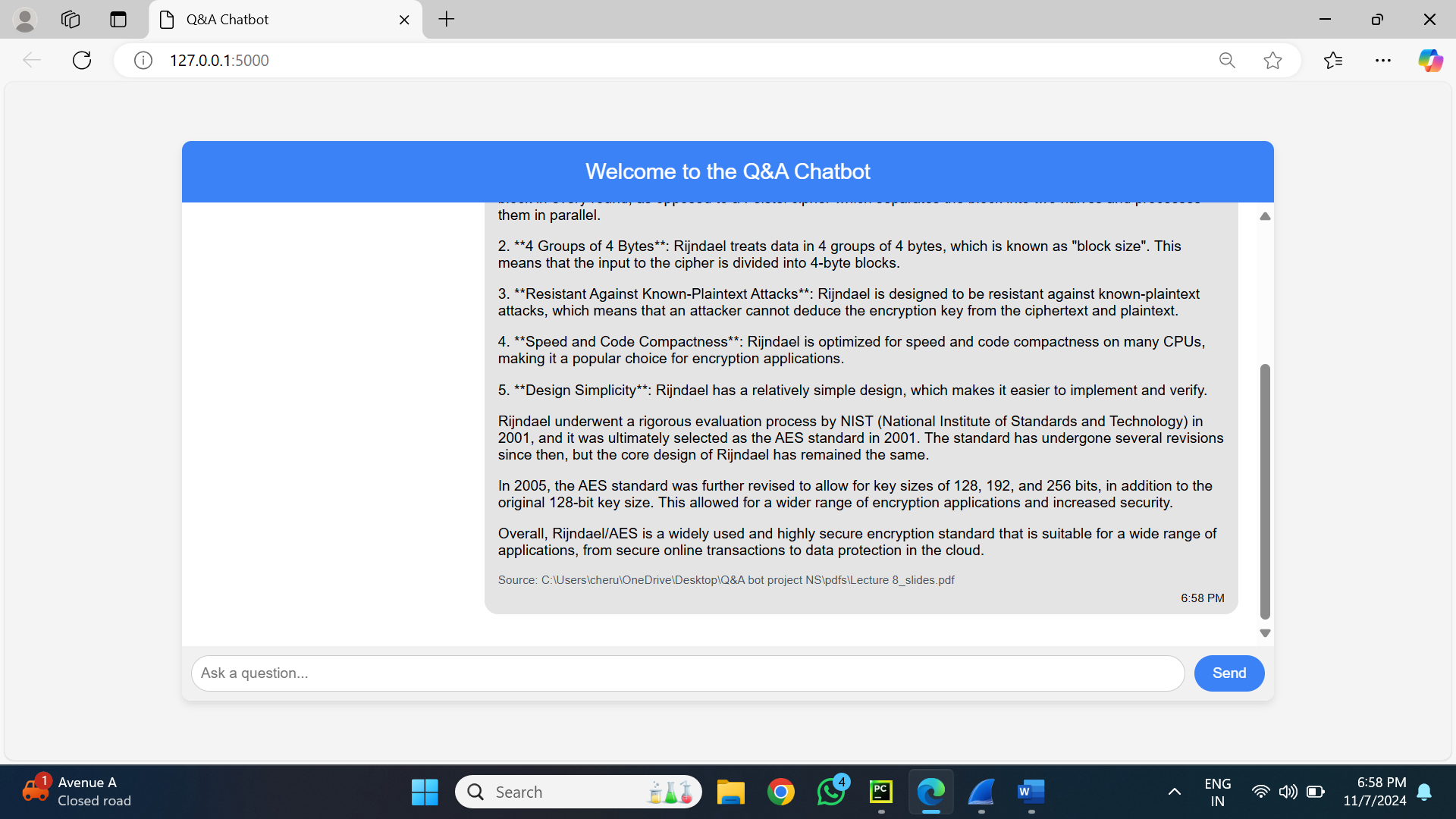
**Mapped Response (Packet 11):**

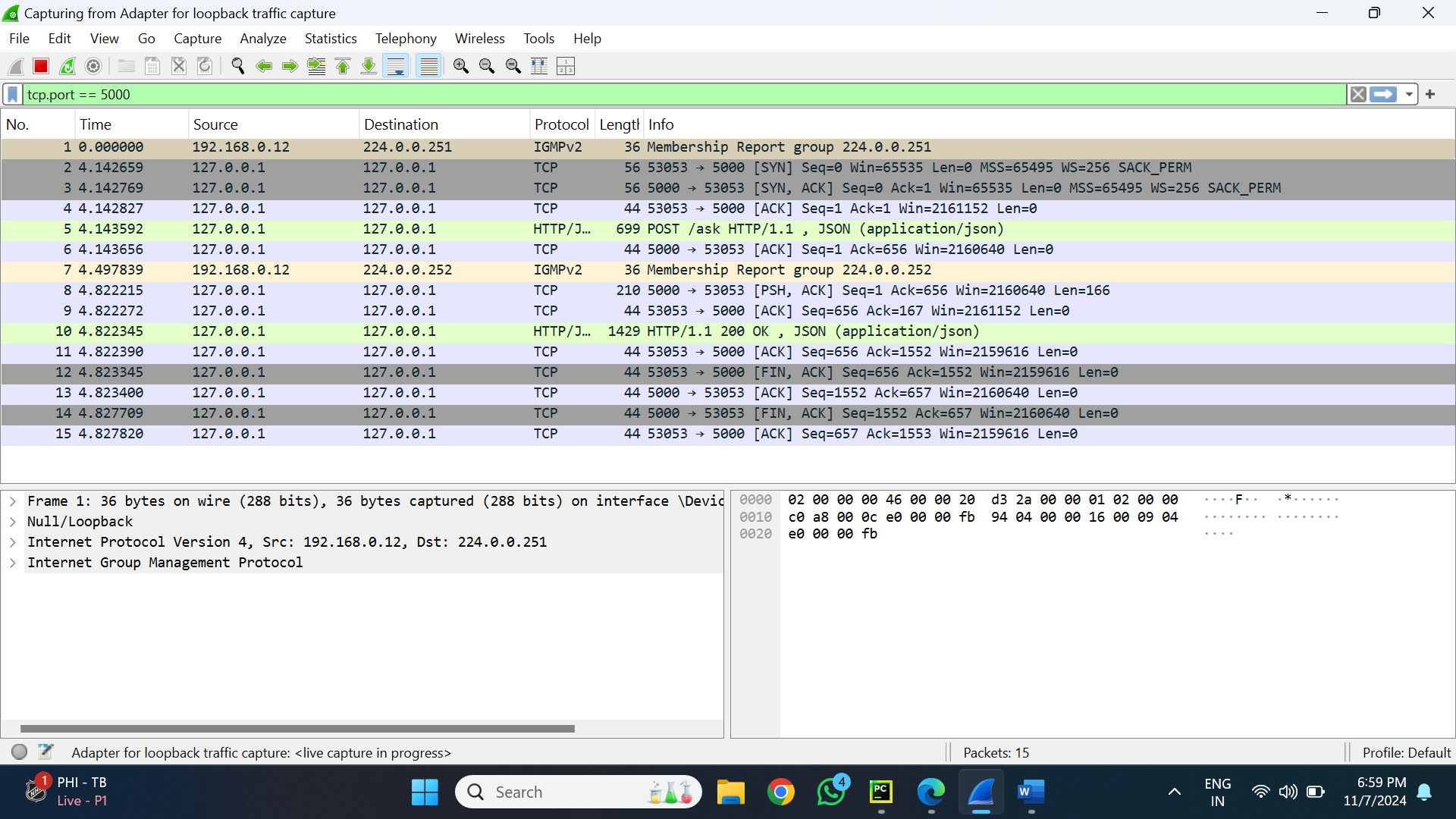
The **answer** field in the JSON payload contains the server's response, which addresses the query in a structured manner.

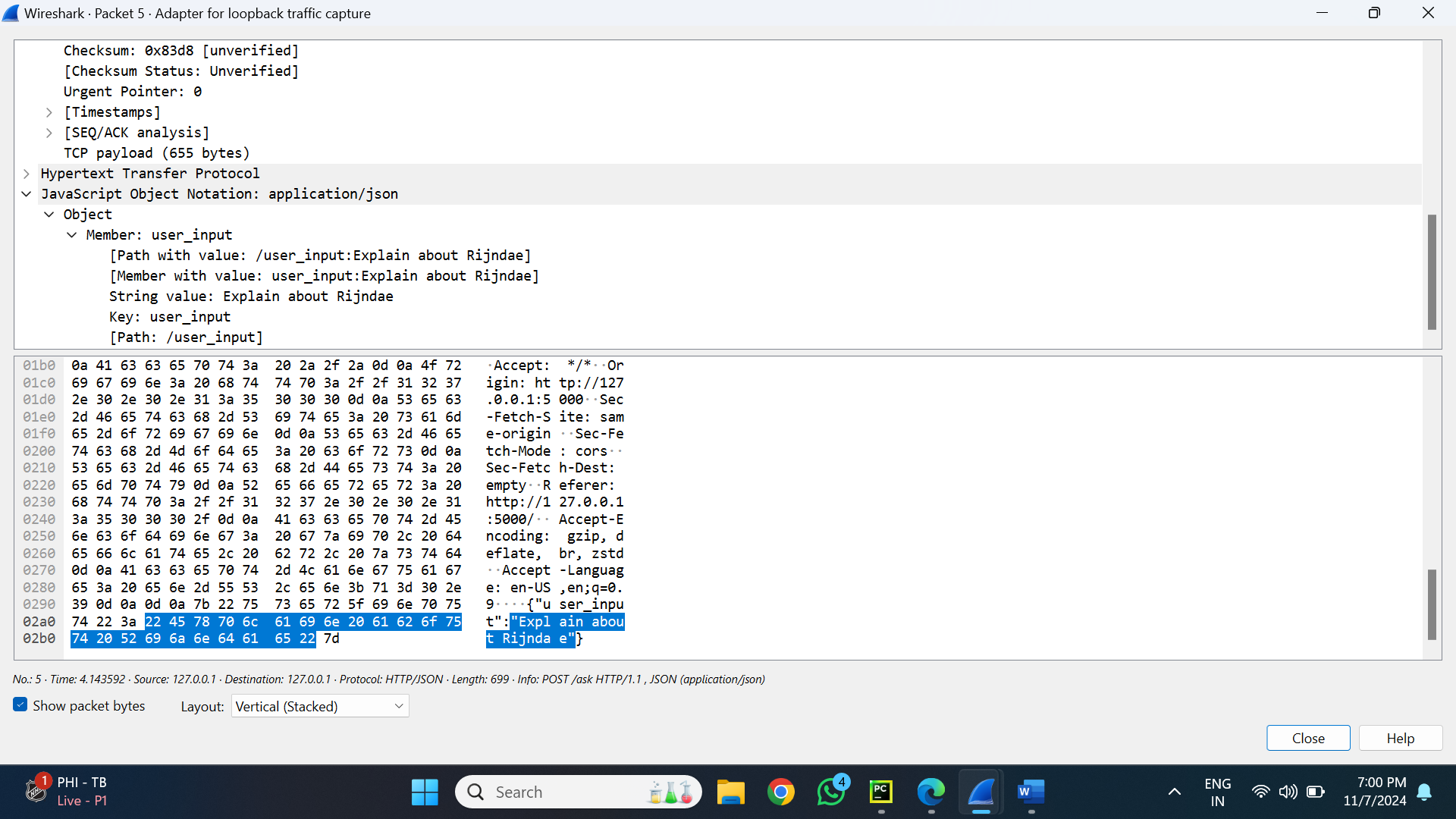
**Summary of Mapping:** The prompt has been summarized in the **user\_input** field of the request payload, processed by the backend server, and returned as a response in the **answer** field.

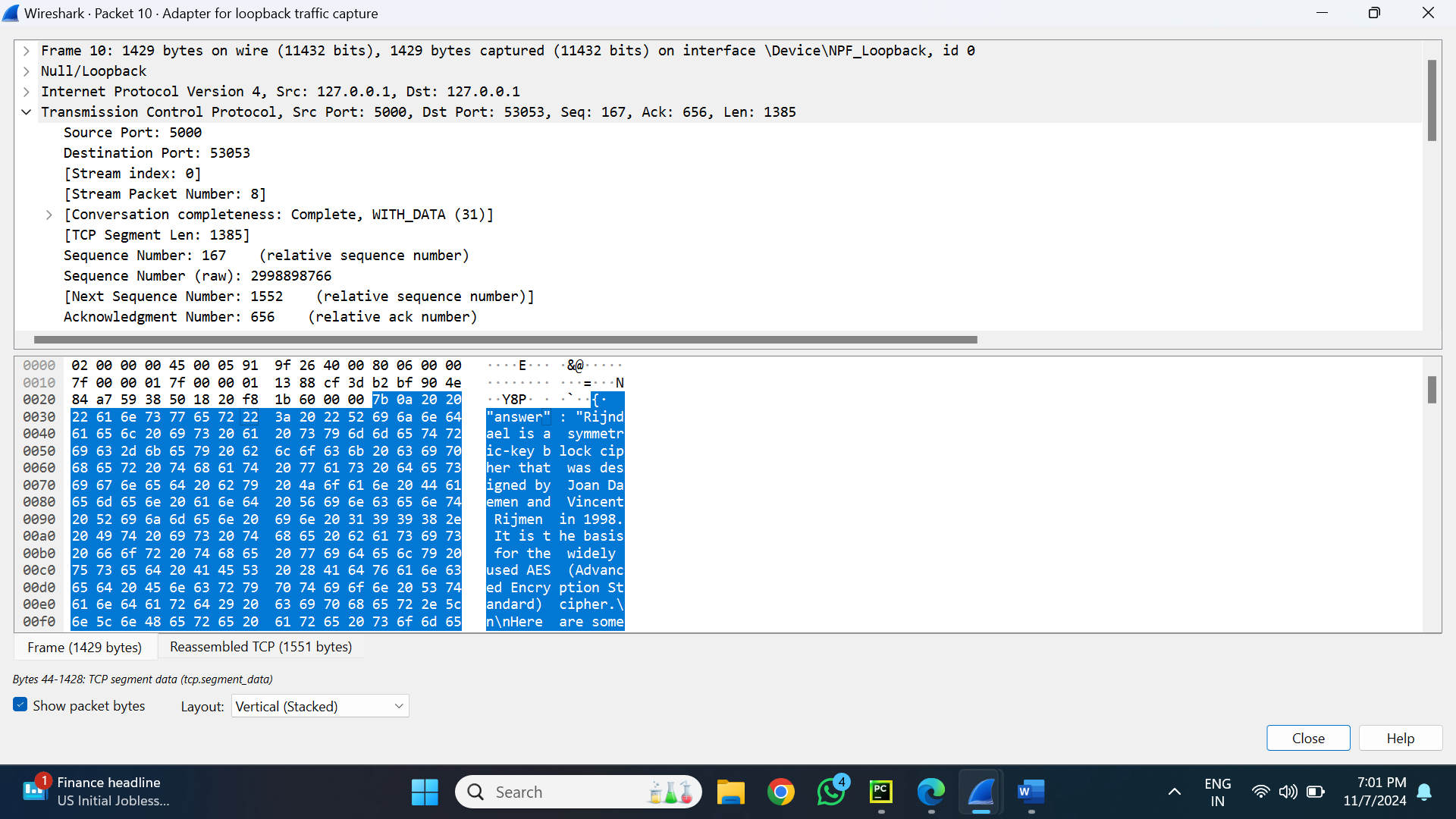
* **Prompt 4**











**Overview of the Prompt 4**

For this prompt a chatbot interface's interactions with a backend LLM server for a particular prompt must be recorded and analysed. This article describes how the user's prompt (Step 1) and the network data recorded during the process (Step 4) are mapped. Wireshark was used to capture the data flow.

**Step 1: User's Prompt** The user initiated the query via the chatbot interface with the prompt:  
 **"Explain about Rijndael."**  
This input is the starting point, which gets transmitted to the LLM server via an API request.

**Step 2: Request Data Capture (Wireshark: Packet 5)**

**Network Details:**

* **Source:** 127.0.0.1 (localhost)
* **Destination:** 127.0.0.1 (localhost)
* **Source Port:** 53053
* **Destination Port:** 5000

**Request Payload:** The payload transmitted contains the following JSON structure:

{

"user\_input": "Explain about Rijndael"

}

**Key Observations in Request:**

* The **user\_input** field is directly mapped to the user's original query.
* The data travels via a TCP connection within the same machine.

**Step 3: Response Data Capture (Wireshark: Packet 10)**

**Network Details:**

* **Source:** 127.0.0.1 (localhost)
* **Destination:** 127.0.0.1 (localhost)
* **Source Port:** 5000
* **Destination Port:** 53053

**Response Payload:** The response JSON received from the backend server is:

{

"answer": "Rijndael is a block cipher that was designed by Joan Daemen and Vincent Rijmen in 1998. It is also known as AES (Advanced Encryption Standard) due to its adoption as a widely accepted standard for symmetric-key encryption.\n\nHere are some key characteristics of Rijndael:\n1. \*\*Iterative Cipher\*\*: Rijndael is an iterative rather than Feistel cipher, meaning that it processes the entire block in every round, as opposed to a Feistel cipher which separates the block into two halves and processes them in parallel.\n2. \*\*4 Groups of 4 Bytes\*\*: Rijndael treats data in 4 groups of 4 bytes, which is known as 'block size.' This means that the input to the cipher is divided into 16-byte blocks.\n3. \*\*Resistant Against Known-Plaintext Attacks\*\*: Rijndael is designed to be resistant against known-plaintext attacks, which means that an attacker cannot deduce the encryption key from the ciphertext and plaintext.\n4. \*\*Speed and Code Compactness\*\*: Rijndael is optimized for speed and code compactness on many CPUs, making it a popular choice for encryption applications.\n5. \*\*Design Simplicity\*\*: Rijndael has a relatively simple design, which makes it easier to implement and verify.\n\nRijndael underwent a rigorous evaluation process by NIST in 2001 and was ultimately selected as the AES standard. The AES standard was further revised in 2005 to allow for key sizes of 128, 192, and 256 bits, in addition to the original 128-bit key size. Overall, Rijndael/AES is a widely used encryption standard suitable for secure online transactions and data protection in the cloud."

}

**Key Observations in Response:**

* Rijndael, its features, and its adoption as the Advanced Encryption Standard (AES) are all thoroughly explained in the response area.
* The response explains its iterative structure, block size, resistance to attacks, speed, and simplicity.

**Step 4: Mapping Between Prompt and Trace Data**

**Prompt (User Input):** The user's original query:  
 **"Explain about Rijndael."**

**Mapped Request (Packet 5):**

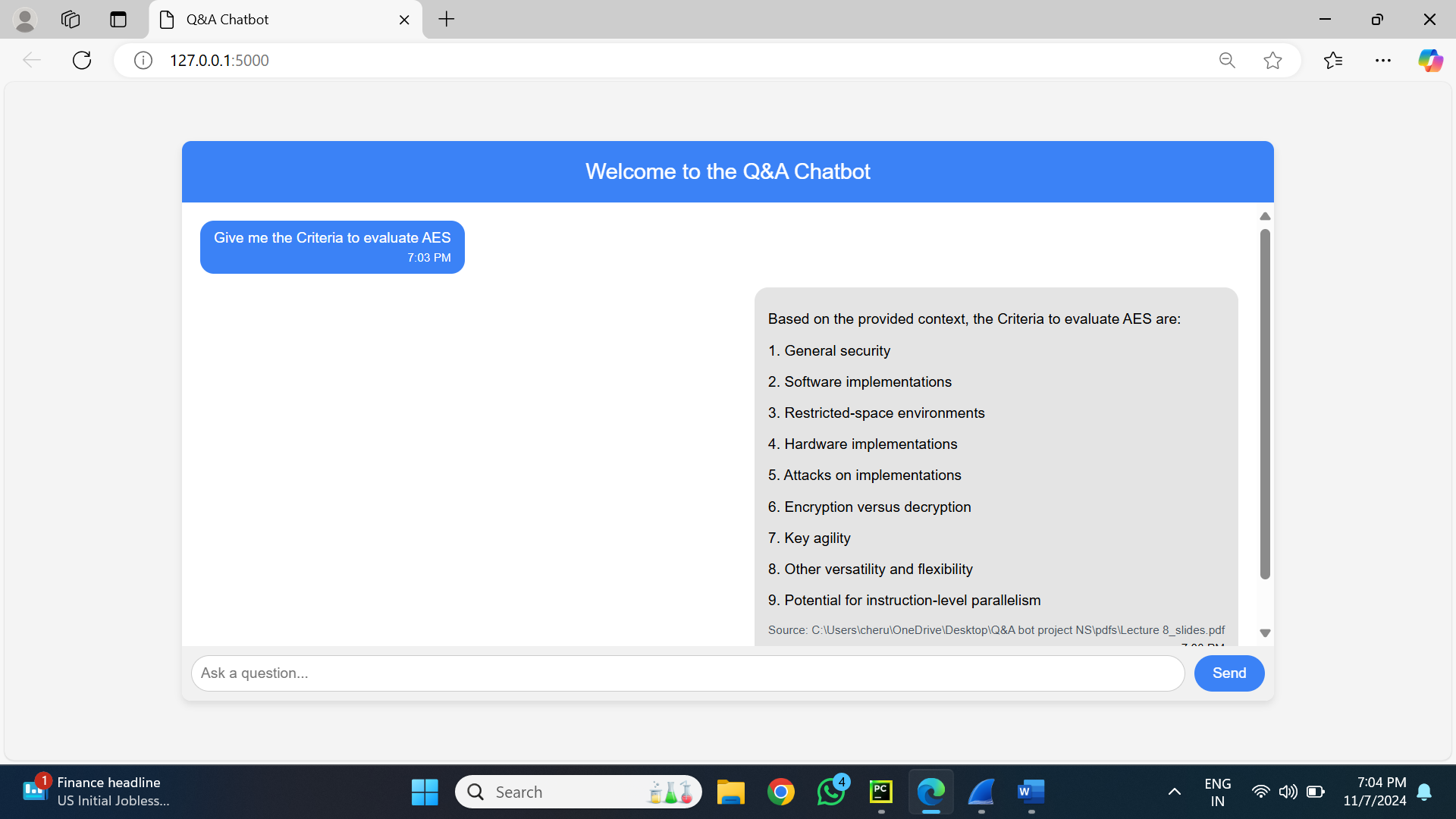
The **user\_input** field in the JSON payload contains the exact text of the prompt.

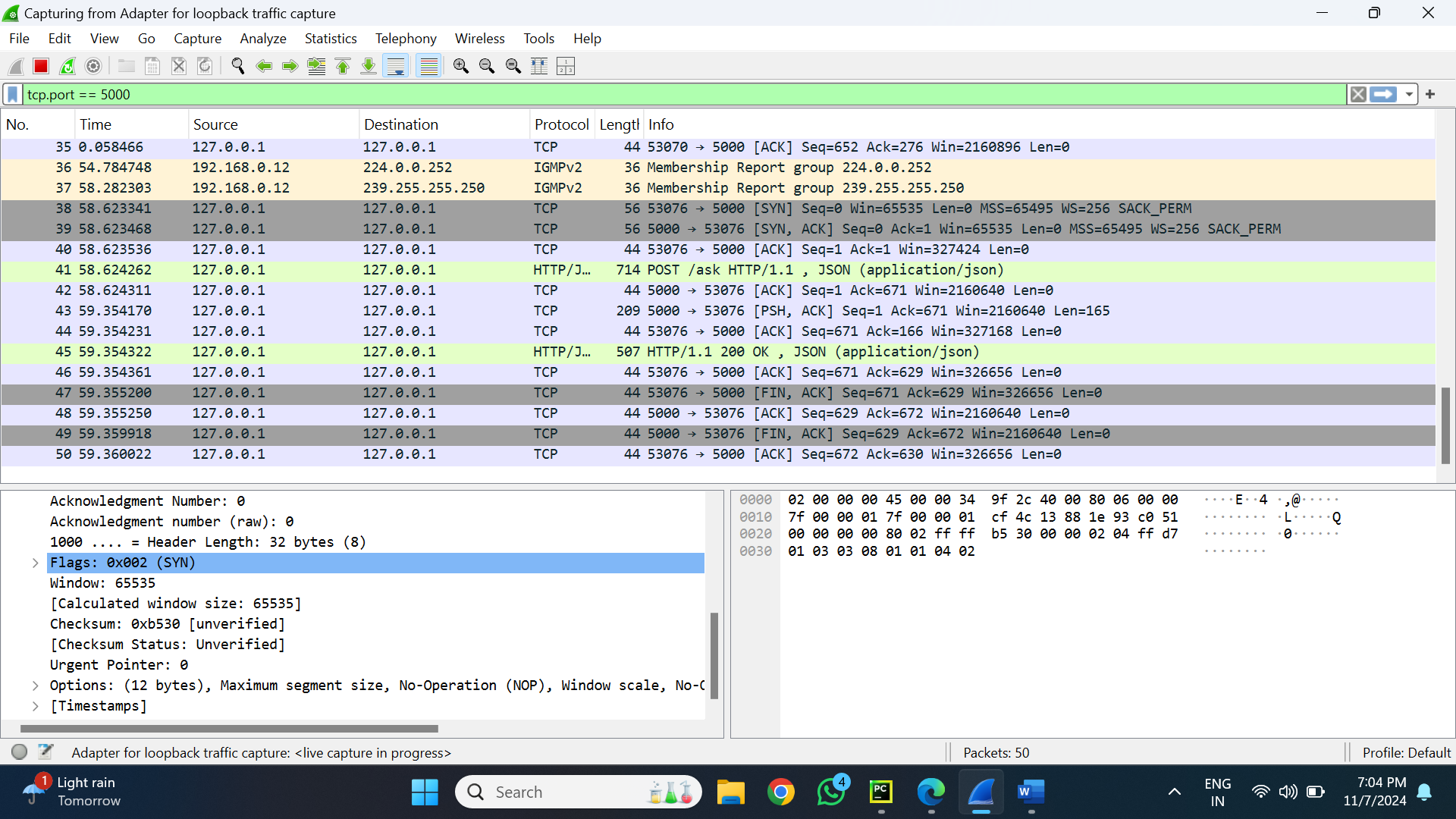
**Mapped Response (Packet 10):**

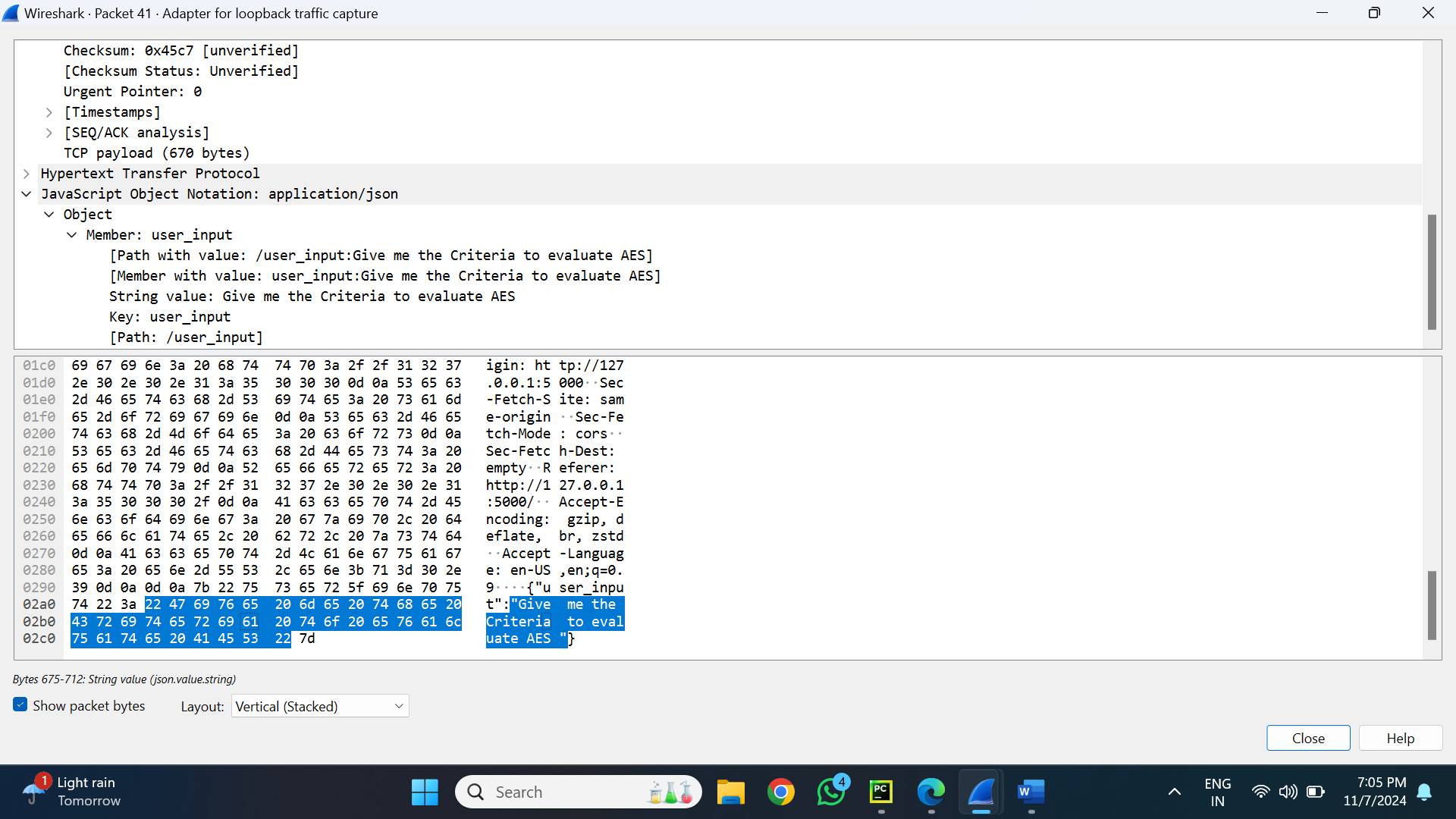
The **answer** field in the JSON payload contains the server's response, which addresses the query in a structured manner.

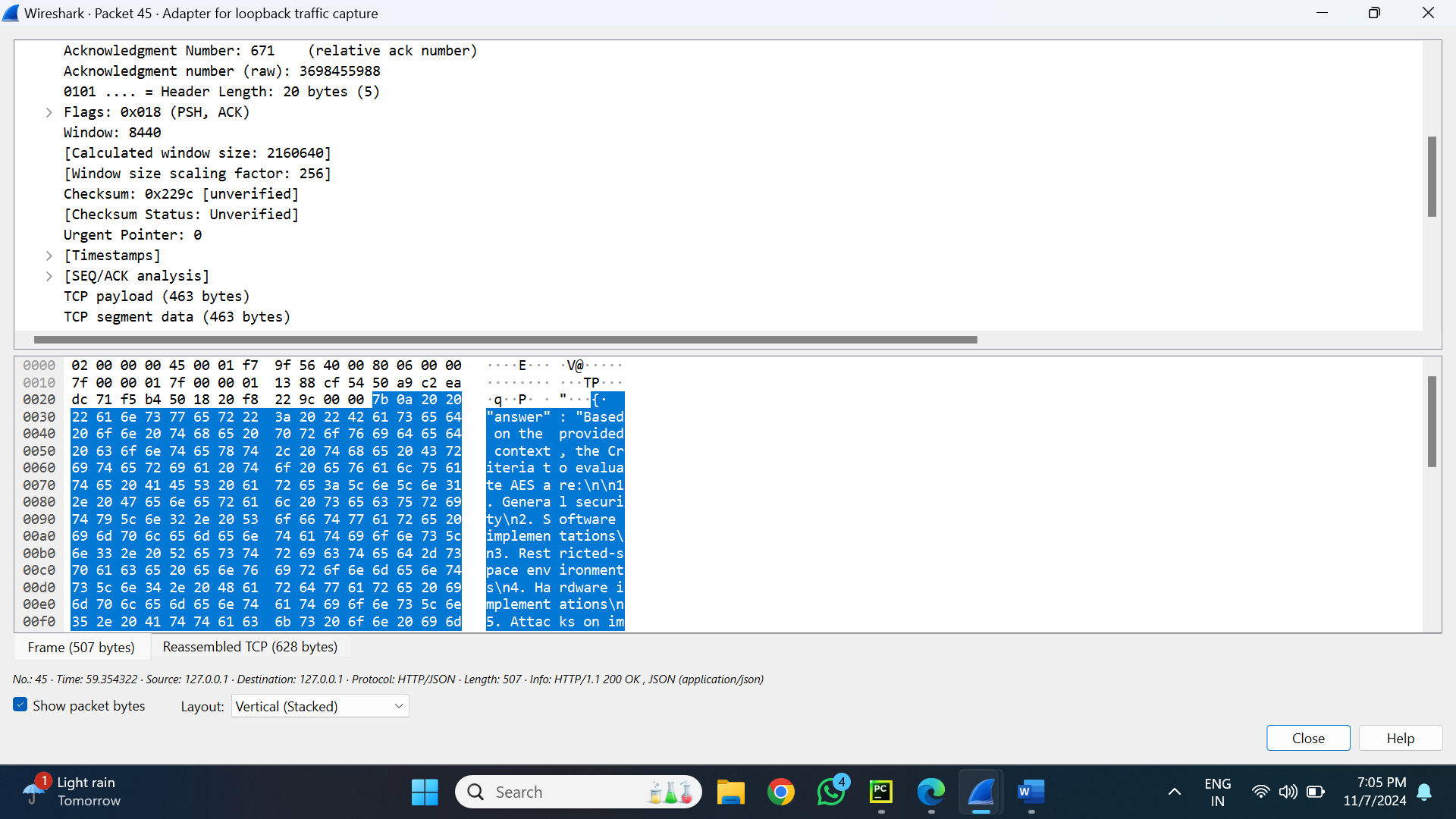
**Summary of Mapping:** The prompt is summarized in the **user\_input** field of the request payload, processed by the backend server, and returned as a response in the **answer** field.

* **Prompt 5**









**Overview of the Prompt 5**

For this prompt a chatbot interface's interactions with a backend LLM server for a particular prompt must be recorded and analysed. This article describes how the user's prompt (Step 1) and the network data recorded during the process (Step 4) are mapped. Wireshark was used to capture the data flow.

**Step 1: User's Prompt** The user initiated the query via the chatbot interface with the prompt:  
 **"Give me the Criteria to evaluate AES."**  
This input is the starting point, which gets transmitted to the LLM server via an API request.

**Step 2: Request Data Capture (Wireshark: Packet 41)**

**Network Details:**

* **Source:** 127.0.0.1 (localhost)
* **Destination:** 127.0.0.1 (localhost)
* **Source Port:** 53076
* **Destination Port:** 5000

**Request Payload:** The payload transmitted contains the following JSON structure:

{ "user\_input": "Give me the Criteria to evaluate AES" }

**Key Observations in Request:**

* The **user\_input** field is directly mapped to the user's original query.
* The data travels via a TCP connection within the same machine.

**Step 3: Response Data Capture (Wireshark: Packet 45)**

**Network Details:**

* **Source:** 127.0.0.1 (localhost)
* **Destination:** 127.0.0.1 (localhost)
* **Source Port:** 5000
* **Destination Port:** 53076

**Response Payload:** The response JSON received from the backend server is:

{

"answer": "Based on the provided context, the Criteria to evaluate AES are:\n1. General security\n2. Software implementations\n3. Restricted-space environments\n4. Hardware implementations\n5. Attacks on implementations\n6. Encryption versus decryption\n7. Key agility\n8. Other versatility and flexibility\n9. Potential for instruction-level parallelism"

}

**Key Observations in Response:**

• A thorough list of evaluation criteria for AES is provided in the answer box, encompassing topics like security, implementations, adaptability, and flexibility.   
•The answer is clearly organized into a numbered list that is directly related to the question.

**Step 4: Mapping Between Prompt and Trace Data**

**Prompt (User Input):** The user's original query:  
 **"Give me the Criteria to evaluate AES."**

**Mapped Request (Packet 41):**

The **user\_input** field in the JSON payload contains the exact text of the prompt.

**Mapped Response (Packet 45):**

The **answer** field in the JSON payload contains the server's response, which addresses the query in a structured manner.

**Summary of Mapping:** The prompt is summarized in the **user\_input** field of the request payload, processed by the backend server, and returned as a response in the **answer** field.