**PDCP Assignments**

**Question 1 - What is the role of PDCP in the LTE and 5G networks?**

**Answer –**

The Packet Data Convergence Protocol (PDCP) plays a crucial role in both LTE (Long-Term Evolution) and 5G networks. It operates at the layer 2.5 level of the network protocol stack and is responsible for several key functions:

**In LTE (4G):**

1. **Header Compression:** PDCP compresses the headers of IP packets to reduce overhead and improve efficiency. This is particularly important for enhancing throughput and reducing latency, especially in environments with limited bandwidth.
2. **Ciphering and Deciphering:** It handles encryption and decryption of user data. This ensures the confidentiality and integrity of the data being transmitted over the network.
3. **Data Integrity:** PDCP provides data integrity protection by including mechanisms to detect and correct errors that may occur during transmission.
4. **Reordering:** It reorders packets to ensure that data is delivered in the correct sequence, as IP packets can arrive out of order due to varying network paths and delays.
5. **Data Forwarding:** PDCP is responsible for transferring data between the eNodeB (evolved NodeB) and the UE (User Equipment) in LTE networks.

**In 5G:**

The PDCP layer in 5G retains similar functions to its LTE counterpart but with some enhancements to support the new features and requirements of 5G:

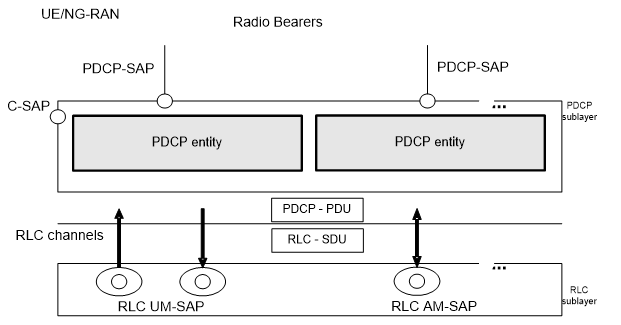
1. **Header Compression:** PDCP continues to compress headers, but in 5G, it also supports more advanced compression techniques, such as the new ROHC (Robust Header Compression) mechanism for further efficiency.
2. **Ciphering and Deciphering:** PDCP maintains encryption and decryption functionality, ensuring secure communication. In 5G, the encryption algorithms are updated to meet the higher security standards.
3. **Data Integrity:** Data integrity protection remains a key function, with enhancements to improve error detection and correction in the more complex 5G environment.
4. **Reordering:** Packet reordering continues to ensure that data is delivered in the correct sequence. This is especially important in 5G due to the higher speeds and more diverse use cases.
5. **Data Forwarding:** PDCP is responsible for transferring data in the 5G system, supporting both the enhanced Mobile Broadband (eMBB) and the ultra-reliable low-latency communication (URLLC) use cases.
6. **Session Management:** In 5G, PDCP also supports new functionalities related to session management and context handling, which are essential for the more complex 5G architecture involving multiple network slices and types of traffic.

Overall, PDCP is vital for optimizing data transfer, ensuring security, and maintaining the quality of service in both LTE and 5G networks.

**Question 2- explain PDCP Structure and entity in details for each steps?**

**Answer –**

The Packet Data Convergence Protocol (PDCP) is a key protocol in the LTE (Long-Term Evolution) and 5G systems used to enhance the performance of the radio interface. It is responsible for managing the data sent over the air interface between the User Equipment (UE) and the eNodeB (in LTE) or gNodeB (in 5G). The PDCP protocol operates in the Layer 2 of the OSI model, which sits above the RLC (Radio Link Control) layer and below the IP layer.



**1. PDCP Structure**

The PDCP layer is responsible for several key functions:

**a. Header Compression and Decompression**

* **Purpose:** To reduce the overhead of header information in transmitted data, enhancing the efficiency of the radio link.
* **Example:** In LTE, it uses Robust Header Compression (RoHC) to compress headers of IP packets, which is essential for reducing the size of headers and thus saving bandwidth.

**b. Data Integrity Protection (DIP)**

* **Purpose:** To ensure the integrity of the data transmitted over the radio interface, PDCP provides mechanisms to detect and correct errors.
* **Example:** In LTE, PDCP provides integrity protection for signaling messages to ensure that they are not tampered with during transmission.

**c. Ciphering and Deciphering**

* **Purpose:** To secure data by encrypting it before transmission and decrypting it upon reception.
* **Example:** LTE uses encryption algorithms like AES (Advanced Encryption Standard) to protect the confidentiality of user data.

**d. Sequence Numbering and Reordering**

* **Purpose:** To ensure that data packets are delivered in the correct order and to manage the sequence of packets.
* **Example:** PDCP uses sequence numbers to detect out-of-order packets and reorder them accordingly.

**e. Delivery of User Data and Signaling Data**

* **Purpose:** To manage the delivery of both user plane data (e.g., internet traffic) and control plane data (e.g., signaling messages).
* **Example:** PDCP handles the data transfer for applications and services running on the UE.

**2. PDCP Entities**

**a. PDCP Entity in the UE (User Equipment)**

* **Function:** The PDCP entity in the UE is responsible for compressing and encrypting outgoing data before it is sent over the air interface. It also decompresses and decrypts incoming data received from the network.
* **Details:**
  + **Compression/Decompression:** Applies header compression to outgoing packets and decompresses incoming packets.
  + **Ciphering/Deciphering:** Encrypts outgoing data packets and decrypts incoming data packets.
  + **Reordering:** Handles the reordering of packets that may arrive out of sequence.

**b. PDCP Entity in the eNodeB/gNodeB (Network Side)**

* **Function:** The PDCP entity in the base station (eNodeB in LTE, gNodeB in 5G) performs similar functions to the UE’s PDCP entity, but in the reverse direction.
* **Details:**
  + **Compression/Decompression:** Applies header compression to incoming packets from the core network and decompresses outgoing packets destined for the UE.
  + **Ciphering/Deciphering:** Decrypts incoming data packets and encrypts outgoing data packets.
  + **Reordering:** Ensures that data packets are delivered in the correct order to the core network.

**3. PDCP Steps in Data Transfer**

**a. Data Transfer from UE to eNodeB/gNodeB**

1. **Data Preparation:** The PDCP entity in the UE receives data from the upper layers (e.g., RLC layer).
2. **Header Compression:** Compresses the IP headers using RoHC.
3. **Ciphering:** Encrypts the compressed data.
4. **Segmentation:** If necessary, segments the data into smaller packets.
5. **Transmission:** Sends the encrypted and compressed data packets over the air interface to the eNodeB/gNodeB.

**b. Data Transfer from eNodeB/gNodeB to Core Network**

1. **Reception:** The eNodeB/gNodeB receives the data packets from the UE.
2. **Deciphering:** Decrypts the data packets.
3. **Header Decompression:** Decompresses the IP headers.
4. **Delivery:** Forwards the data to the appropriate network entity (e.g., S-GW in LTE, UPF in 5G).

**c. Data Transfer from Core Network to eNodeB/gNodeB**

1. **Reception:** The eNodeB/gNodeB receives data packets from the core network.
2. **Ciphering:** Encrypts the data.
3. **Header Compression:** Compresses the IP headers.
4. **Segmentation:** If necessary, segments the data into smaller packets.
5. **Transmission:** Sends the encrypted and compressed data packets over the air interface to the UE.

**d. Data Transfer from eNodeB/gNodeB to UE**

1. **Reception:** The UE receives the data packets from the eNodeB/gNodeB.
2. **Deciphering:** Decrypts the data packets.
3. **Header Decompression:** Decompresses the IP headers.
4. **Delivery:** Delivers the data to the upper layers (e.g., application layer) in the UE.

**Question 3 - Describe how PDCP provides confidentiality to the data transmitted over the air interface?**

**Answer –**

The Packet Data Convergence Protocol (PDCP) is an essential component in the LTE (Long-Term Evolution) and 5G cellular networks, handling the data link layer responsibilities for both user plane and control plane data. One of its critical functions is to ensure data confidentiality over the air interface. Here’s how PDCP provides confidentiality to the transmitted data:

**1. Encryption of User Data**

PDCP is responsible for encrypting the user data (payload) to protect it from unauthorized access while it is transmitted over the air interface. The encryption process involves the following steps:

* **Key Generation:** During the initial network attach or during key renewal procedures, encryption keys are generated and exchanged between the user equipment (UE) and the evolved NodeB (eNodeB) or gNodeB (for 5G).
* **Encryption Algorithm:** PDCP uses encryption algorithms specified by the standards, such as AES (Advanced Encryption Standard). The data is encrypted using these algorithms with keys derived from the initial keying material established during the network attach process.
* **Ciphering:** Once the encryption key is established, PDCP applies the ciphering process to the user data packets. This transforms the original data into an unreadable format that can only be decrypted with the correct key.

**2. Integrity Protection**

In addition to encryption, PDCP provides integrity protection to ensure the data's authenticity and to verify that it hasn't been altered during transmission. This process includes:

* **Integrity Key:** Similar to encryption, an integrity key is generated and used to protect the data. The integrity key is part of the security context established during the UE's initial access to the network.
* **Integrity Algorithm:** PDCP uses integrity algorithms, such as the Integrity Check Value (ICV) with algorithms like AES, to create a hash of the data. This hash is transmitted along with the data, allowing the receiver to verify that the data hasn't been tampered with.

**3. Sequence Numbers and Reordering**

To prevent replay attacks and ensure the correct sequencing of data, PDCP includes sequence numbers in the data packets. This helps in detecting and discarding any duplicate or out-of-order packets that might have been intercepted and replayed by an attacker.

**4. Ciphering and Integrity in Transport Channels**

PDCP operates above the Radio Link Control (RLC) layer and is part of the Radio Access Network (RAN) architecture. It interfaces with transport channels, where it handles the encryption and integrity protection before the data is sent over the air interface. This ensures that all data transmitted from the UE to the network, and vice versa, is securely protected.

**Question 4 - Explain the significance of header compression in PDCP?**

**Answer –**

Header compression in the Packet Data Convergence Protocol (PDCP) is a key feature for optimizing network efficiency, especially in mobile networks like LTE and 5G. Here’s why it’s significant:

**1. Reduction of Overhead**

PDCP is responsible for handling data between the user equipment (UE) and the evolved NodeB (eNodeB) or the Next Generation NodeB (gNB) in LTE and 5G networks, respectively. Data packets include both payload and header information. Header compression reduces the size of headers, which are the metadata associated with each packet, thus minimizing the overall amount of data that needs to be transmitted.

**2. Efficient Use of Bandwidth**

By compressing headers, less bandwidth is used for transmitting the same amount of user data. This is particularly important in wireless networks where bandwidth is often limited. Efficient header compression allows more user data to be transmitted within the same bandwidth, improving the overall data throughput and network efficiency.

**3. Improved Performance**

Reducing header size decreases the amount of data that needs to be processed and transmitted. This leads to reduced transmission delays and can improve the performance of applications that are sensitive to latency, such as real-time video streaming or VoIP.

**4. Reduced Power Consumption**

In mobile devices, transmitting less data not only improves network performance but also reduces power consumption. This is beneficial for extending the battery life of mobile devices, which is a critical factor for user satisfaction.

**5. Optimization of Network Resources**

Header compression helps in optimizing network resources by reducing the amount of data that needs to be handled by network nodes. This can lead to better resource utilization and can help in managing network congestion more effectively.

**6. Support for Different Protocols**

In PDCP, header compression is particularly relevant for protocols like HTTP and TCP, which can have large headers. PDCP uses techniques like Robust Header Compression (RoHC) to efficiently compress these headers, ensuring that the protocol overhead does not significantly impact the performance of the data session.

**Question 5 - How does PDCP handle data integrity for control plane messages?**

**Answer -** The Packet Data Convergence Protocol (PDCP) is a key component in the LTE (Long-Term Evolution) and 5G wireless communication systems, and it plays a crucial role in handling both user plane and control plane data. For control plane messages, PDCP ensures data integrity through several mechanisms:

1. **Integrity Protection:** PDCP uses integrity protection to ensure that control plane messages are not tampered with during transmission. This involves using cryptographic algorithms to create a message authentication code (MAC) that is appended to the control plane data. Upon receipt, the receiver uses the same algorithm and a shared secret key to verify the MAC. If the computed MAC matches the received MAC, the message is considered intact and unaltered. This process helps in detecting any unauthorized modifications.
2. **Encryption:** In addition to integrity protection, control plane messages can also be encrypted. Encryption ensures that even if the messages are intercepted, their contents remain confidential. The encryption process transforms the data into a format that cannot be understood without the appropriate decryption key.
3. **Sequence Numbering**: PDCP uses sequence numbers to track and order control plane messages. This helps in detecting lost or out-of-order packets and ensures that messages are processed in the correct sequence. The sequence number is used in conjunction with integrity protection to verify that all messages are received as intended.
4. **Reordering and Retransmission:** PDCP also implements mechanisms for reordering and retransmitting control plane messages if they are lost or delivered out of sequence. This ensures that the control plane protocol operates correctly even in the presence of network issues.

**Question 6 - Discuss the sequence numbering mechanism in PDCP and its importance for data transmission.**

**Answer -**

The Packet Data Convergence Protocol (PDCP) is a key layer in the LTE (Long-Term Evolution) and 5G mobile network stack that handles the data transfer between the user equipment (UE) and the network. One of the essential features of PDCP is its sequence numbering mechanism, which plays a crucial role in ensuring reliable data transmission.

**Sequence Numbering in PDCP**

In PDCP, sequence numbering is used to track and manage the data packets as they are transmitted between the UE and the evolved NodeB (eNB) or gNB (in 5G networks). The main elements of sequence numbering in PDCP are:

1. **Sequence Number (SN):**
   * PDCP uses a sequence number for each data packet it transmits. This sequence number is crucial for maintaining the correct order of packets and for detecting any packet loss or duplication.
   * The sequence number is typically a 12-bit field, which means it can range from 0 to 4095. This range allows for a sufficient number of packets to be managed before sequence numbers wrap around.
2. **Protocol Data Units (PDUs):**
   * PDCP Protocol Data Units (PDUs) are the packets that PDCP processes. Each PDU is assigned a unique sequence number, which helps the receiver correctly reassemble the data in the proper order.
3. **Reordering and Duplication Detection:**
   * At the receiver end, the sequence number allows the PDCP layer to reorder packets if they arrive out of sequence. This is important because packets might take different routes through the network or might be subject to varying network conditions.
   * The sequence number also helps in detecting duplicate packets. If a packet with a sequence number that has already been received arrives, the receiver can discard it as a duplicate.

**Importance of Sequence Numbering in Data Transmission**

1. **Data Integrity and Order:**
   * Sequence numbering ensures that data is delivered in the same order as it was sent. This is crucial for maintaining the integrity of the data stream, especially for applications that require ordered data, such as video streaming or voice calls.
2. **Error Handling:**
   * Sequence numbers enable the detection of lost packets. If a packet is missing, the receiver can identify the gap in the sequence and request retransmission if necessary.
3. **Efficient Use of Resources:**
   * By tracking which packets have been acknowledged and which have not, the PDCP layer can optimize retransmission and avoid unnecessary network congestion.
4. **Quality of Service (QoS):**
   * Accurate sequence numbering contributes to maintaining the quality of service by ensuring that data is consistently delivered in the correct order and without loss. This is particularly important for maintaining low latency and high reliability in real-time applications.

**Question 7 - Describe the process and purpose of ROHC within PDCP.  
How does PDCP handle duplicate data packets in scenarios involving dual connectivity?**

**Answer –**

**ROHC (Robust Header Compression) in PDCP**

**Process and Purpose:**

1. **Overview:**
   * ROHC (Robust Header Compression) is a technique used in mobile networks to compress headers of IP packets, reducing the overhead and improving efficiency.
   * In the context of PDCP (Packet Data Convergence Protocol), ROHC is used to compress headers of IP packets transmitted over the radio interface. This is particularly important for optimizing bandwidth and improving performance in cellular networks.
2. **Compression Process:**
   * **Initialization:**
     + The ROHC process begins with the establishment of a context between the sender and receiver. This context holds information about the state of the compression process.
   * **Header Compression:**
     + ROHC compresses the headers of IP packets by removing redundancy. For example, it can compress IPv4 or IPv6 headers, as well as UDP or TCP headers, by using short identifiers or context-based encoding.
   * **Packet Transmission:**
     + The compressed headers are then transmitted along with the data payload. The compressed headers take up less space compared to their uncompressed versions.
   * **Decompression:**
     + On the receiving end, the compressed headers are decompressed using the shared context. The decompressed headers are reconstructed to their original form so that the data can be properly processed by higher layers of the protocol stack.
3. **Purpose:**
   * **Efficiency:** Reduces the amount of data that needs to be transmitted, thereby saving bandwidth and reducing latency.
   * **Improved Throughput:** By compressing headers, more data can be transmitted in the same amount of radio resources, improving overall throughput.
   * **Enhanced Performance:** Helps in handling high volumes of small packets efficiently, which is common in scenarios such as VoIP or real-time applications.

**Handling Duplicate Data Packets in Dual Connectivity**

**Scenario:**

* **Dual Connectivity (DC):** In modern cellular networks, dual connectivity allows a user equipment (UE) to connect simultaneously to multiple base stations (e.g., LTE and NR/5G). This can lead to potential challenges with packet duplication.

**PDCP Handling of Duplicate Packets:**

1. **Sequence Numbers:**
   * **Packet Identification:** PDCP uses sequence numbers to uniquely identify packets. Each data packet sent over the network is tagged with a sequence number that helps in tracking the order and detecting duplicates.
   * **Duplicate Detection:** When the PDCP layer receives packets, it checks the sequence numbers to determine if a packet is a duplicate. If the sequence number indicates that the packet has already been received, it is discarded to avoid processing the same data multiple times.
2. **Reordering:**
   * **Packet Reordering:** If packets arrive out of order due to the simultaneous connections, PDCP can reorder them based on their sequence numbers to ensure data is delivered in the correct order to higher layers.
3. **Buffering:**
   * **Buffer Management:** PDCP maintains buffers to temporarily store incoming packets until they can be properly processed. This buffering helps in managing out-of-order packets and ensuring that duplicates are handled correctly.
4. **Acknowledgements:**
   * **Acknowledgement Handling:** PDCP also uses acknowledgements to confirm the successful reception of packets. If a packet is acknowledged as received, subsequent duplicates are identified and discarded based on this acknowledgment information.

**Question 8 - Explain the role of PDCP in uplink and downlink data transfer. What differences, if any, exist between these processes?**

**Answer –**

The Packet Data Convergence Protocol (PDCP) plays a crucial role in both uplink and downlink data transfer within the LTE (Long-Term Evolution) and 5G systems. It operates in the radio access network (RAN) layer and is responsible for various tasks that facilitate efficient and reliable data communication.

**PDCP in Uplink and Downlink Data Transfer:**

**Downlink Data Transfer:**

1. **Data Header Compression:** In the downlink direction (from the eNodeB/gNodeB to the User Equipment or UE), PDCP compresses the headers of the data packets. This is known as Header Compression (RoHC - Robust Header Compression). Compressing the headers reduces the amount of redundant information transmitted over the radio interface, which helps to optimize the use of radio resources.
2. **Encryption:** PDCP is responsible for encrypting the user data to ensure confidentiality and security. This is done using the encryption keys derived from the higher layers.
3. **Decompression and Decryption:** Once the data reaches the UE, PDCP decompresses the headers and decrypts the user data, making it available for higher layers to process.
4. **In-sequence Delivery:** PDCP ensures that data packets are delivered in the correct order. It handles reordering of packets if they arrive out of sequence.
5. **Data Integrity:** PDCP provides mechanisms to check the integrity of the data and perform retransmissions if necessary (though in LTE, this is less common since higher layers manage retransmissions).

**Uplink Data Transfer:**

1. **Data Header Compression:** Similar to downlink, PDCP compresses the headers of data packets sent from the UE to the eNodeB/gNodeB, which helps to optimize the use of the radio interface.
2. **Encryption:** PDCP is responsible for encrypting user data before transmission. This ensures that sensitive information is protected while being sent over the radio interface.
3. **Decompression and Decryption:** On the receiving side at the eNodeB/gNodeB, PDCP decompresses the packet headers and decrypts the data, making it available for further processing in the network.
4. **In-sequence Delivery:** PDCP handles the reordering of packets in the uplink direction as well. It ensures that data packets are delivered to the higher layers in the correct order.
5. **Data Integrity:** PDCP ensures that the data has not been corrupted during transmission by using integrity checks. If needed, it can request retransmissions.

**Differences in Uplink and Downlink Processes:**

* **Compression and Decompression:** While the compression and decompression of headers are performed in both directions, the specific algorithms and mechanisms might differ slightly between uplink and downlink due to differences in traffic patterns and requirements.
* **Encryption and Decryption:** The encryption and decryption processes involve different keys and may follow different protocols depending on the direction of data flow. For uplink, encryption is done by the UE, and for downlink, it's done by the eNodeB/gNodeB.
* **Retransmission and Integrity Checks:** Although PDCP handles integrity checks and in-sequence delivery in both directions, the actual mechanisms for retransmissions and handling errors might differ. For instance, in LTE, retransmissions are typically handled by the hybrid automatic repeat request (HARQ) layer, but PDCP can be involved in managing data integrity.

**Code 1 - Implement a C++ function that simulates the ciphering process in PDCP for a given payload. Use a simple XOR cipher for demonstration purposes.**

//Code Started

#include <iostream>

#include <string>

// Function to apply XOR ciphering

std::string xorCipher(const std::string &payload, char key) {

std::string cipheredPayload;

for (char c : payload) {

cipheredPayload += c ^ key; // XOR each character with the key

}

return cipheredPayload;

}

int main() {

// Example payload

std::string payload = "Hello, World!";

char key = 'K'; // Simple XOR key

// Cipher the payload

std::string cipheredPayload = xorCipher(payload, key);

std::cout << "Ciphered Payload: ";

for (char c : cipheredPayload) {

std::cout << std::hex << static\_cast<int>(c) << ' ';

}

std::cout << std::endl;

// Decipher the payload

std::string decipheredPayload = xorCipher(cipheredPayload, key);

std::cout << "Deciphered Payload: " << decipheredPayload << std::endl;

return 0;

}

**Code 2 - Write a C++ function to simulate the integrity protection mechanism in PDCP. Use a basic checksum method where the integrity check value is the sum of all bytes in the payload**

//Code Started

#include <iostream>

#include <vector>

// Function to calculate the checksum of a payload

unsigned int calculateChecksum(const std::vector<unsigned char>& payload) {

unsigned int checksum = 0;

for (unsigned char byte : payload) {

checksum += byte;

}

return checksum;

}

// Function to verify the payload with the checksum

bool verifyPayload(const std::vector<unsigned char>& payload, unsigned int expectedChecksum) {

unsigned int calculatedChecksum = calculateChecksum(payload);

return calculatedChecksum == expectedChecksum;

}

int main() {

// Example payload data

std::vector<unsigned char> payload = {0x01, 0x02, 0x03, 0x04, 0x05};

// Calculate checksum

unsigned int checksum = calculateChecksum(payload);

std::cout << "Calculated Checksum: " << checksum << std::endl;

// Verify payload with the calculated checksum

if (verifyPayload(payload, checksum)) {

std::cout << "Payload integrity verified successfully." << std::endl;

} else {

std::cout << "Payload integrity verification failed." << std::endl;

}

return 0;

}

**Code 3 - Develop a C++ function to simulate ROHC compression for a set of IP headers in PDCP. Assume headers are represented as strings of hexadecimal values.**

//Code Started

#include <iostream>

#include <string>

#include <unordered\_map>

#include <vector>

// Function to compress an IP header (simplified simulation)

std::string compressIPHeader(const std::string& header) {

// Example compression: remove '0x' and compress into a short form

std::string compressed;

for (char ch : header) {

if (ch != '0' && ch != 'x') {

compressed += ch;

}

}

return compressed;

}

// Function to decompress an IP header (simplified simulation)

std::string decompressIPHeader(const std::string& compressedHeader) {

// Example decompression: add '0x' back to the header

std::string decompressed = "0x";

for (char ch : compressedHeader) {

decompressed += ch;

}

return decompressed;

}

// Function to simulate ROHC compression and decompression for a set of IP headers

void simulateROHC(const std::vector<std::string>& headers) {

std::unordered\_map<std::string, std::string> headerMap;

std::cout << "Original Headers:\n";

for (const std::string& header : headers) {

std::cout << header << '\n';

}

std::cout << "\nCompressed Headers:\n";

for (const std::string& header : headers) {

std::string compressed = compressIPHeader(header);

headerMap[compressed] = header;

std::cout << compressed << '\n';

}

std::cout << "\nDecompressed Headers:\n";

for (const auto& entry : headerMap) {

const std::string& compressed = entry.first;

const std::string& original = entry.second;

std::string decompressed = decompressIPHeader(compressed);

std::cout << decompressed << " -> " << original << '\n';

}

}

int main() {

std::vector<std::string> ipHeaders = {

"0x4500003c1c4640004006b1e6c0a000001c0a00002",

"0x4500003c1c4640004006b1e6c0a000001c0a00003"

};

simulateROHC(ipHeaders);

return 0;

}

**Code 4 - Create a C++ simulation that handles the processing of PDCP entities, demonstrating the creation, operation, and deletion of a PDCP entity. Include functionality for adding headers and performing a dummy cipher operation.**

//Code Started

#include <iostream>

#include <vector>

#include <string>

#include <memory>

#include <algorithm> // For std::remove\_if

// Define a PDCP Entity class

class PDCP\_Entity {

public:

PDCP\_Entity(int id) : id(id), header(""), data(""), cipheredData("") {

std::cout << "PDCP Entity " << id << " created." << std::endl;

}

~PDCP\_Entity() {

std::cout << "PDCP Entity " << id << " destroyed." << std::endl;

}

// Add a header to the PDCP entity

void addHeader(const std::string& newHeader) {

header = newHeader;

std::cout << "Header added to PDCP Entity " << id << ": " << header << std::endl;

}

// Add data to the PDCP entity

void addData(const std::string& newData) {

data = newData;

std::cout << "Data added to PDCP Entity " << id << ": " << data << std::endl;

}

// Perform a dummy cipher operation

void performCipherOperation() {

cipheredData = "Ciphered(" + data + ")";

std::cout << "Cipher operation performed on PDCP Entity " << id << ": " << cipheredData << std::endl;

}

// Display current state

void displayState() const {

std::cout << "PDCP Entity " << id << " State:" << std::endl;

std::cout << "Header: " << header << std::endl;

std::cout << "Data: " << data << std::endl;

std::cout << "Ciphered Data: " << cipheredData << std::endl;

}

// Get the ID of the entity

int getID() const {

return id;

}

private:

int id;

std::string header;

std::string data;

std::string cipheredData;

};

// PDCP Manager class to handle entity creation, operation, and deletion

class PDCP\_Manager {

public:

void createEntity(int id) {

entities.push\_back(std::unique\_ptr<PDCP\_Entity>(new PDCP\_Entity(id)));

}

void deleteEntity(int id) {

auto it = std::remove\_if(entities.begin(), entities.end(),

[id](const std::unique\_ptr<PDCP\_Entity>& entity) {

return entity->getID() == id;

});

entities.erase(it, entities.end());

std::cout << "PDCP Entity " << id << " deleted." << std::endl;

}

PDCP\_Entity\* getEntity(int id) {

for (auto& entity : entities) {

if (entity->getID() == id) {

return entity.get();

}

}

return nullptr;

}

private:

std::vector<std::unique\_ptr<PDCP\_Entity>> entities;

};

int main() {

PDCP\_Manager manager;

// Create PDCP entities

manager.createEntity(1);

manager.createEntity(2);

// Get and operate on the first entity

PDCP\_Entity\* entity1 = manager.getEntity(1);

if (entity1) {

entity1->addHeader("Header1");

entity1->addData("SampleData1");

entity1->performCipherOperation();

entity1->displayState();

}

// Get and operate on the second entity

PDCP\_Entity\* entity2 = manager.getEntity(2);

if (entity2) {

entity2->addHeader("Header2");

entity2->addData("SampleData2");

entity2->performCipherOperation();

entity2->displayState();

}

// Delete the entities

manager.deleteEntity(1);

manager.deleteEntity(2);

return 0;

}

**Code 5 - Write the code for PDCP inetgrity process**

//Code Started

**//Ensure OpenSSL is Installed**

//First, ensure you have OpenSSL installed on your system. You can install it using your package manager:

**//On Ubuntu/Debian**:

//sudo apt-get update

//sudo apt-get install libssl-dev

**//Compile with Proper Linker Flags**

//You need to make sure you link against the OpenSSL libraries when compiling your C++ code. Use -lssl and -lcrypto to link the SSL and crypto libraries provided by OpenSSL.

**//g++ -o pdcp\_integrity pdcp\_integrity.cpp -lssl -lcrypto**

#include <iostream>

#include <openssl/hmac.h>

#include <openssl/evp.h>

#include <vector>

#include <iomanip>

#include <sstream>

// Helper function to convert a byte array to a hexadecimal string

std::string bytes\_to\_hex(const std::vector<unsigned char>& bytes) {

std::ostringstream oss;

for (unsigned char c : bytes) {

oss << std::hex << std::setw(2) << std::setfill('0') << static\_cast<int>(c);

}

return oss.str();

}

// Class to handle PDCP integrity protection

class PDCPIntegrity {

public:

PDCPIntegrity(const std::vector<unsigned char>& key) : key\_(key) {}

// Generate an integrity protection tag for the given data

std::vector<unsigned char> generate\_integrity\_protection\_tag(const std::vector<unsigned char>& data) {

unsigned char result[EVP\_MAX\_MD\_SIZE]; // Buffer to hold the HMAC result

unsigned int result\_len = 0;

// Compute HMAC with SHA256

HMAC(EVP\_sha256(), key\_.data(), key\_.size(), data.data(), data.size(), result, &result\_len);

// Return the result as a vector

return std::vector<unsigned char>(result, result + result\_len);

}

// Verify the integrity of the given data with the provided tag

bool verify\_integrity(const std::vector<unsigned char>& data, const std::vector<unsigned char>& tag) {

auto generated\_tag = generate\_integrity\_protection\_tag(data);

return generated\_tag == tag;

}

private:

std::vector<unsigned char> key\_;

};

// Helper function to print a vector of bytes

void print\_bytes(const std::vector<unsigned char>& bytes) {

for (unsigned char c : bytes) {

std::cout << c;

}

std::cout << std::endl;

}

int main() {

// Example key and data

std::vector<unsigned char> key = { 's', 'u', 'p', 'e', 'r', 's', 'e', 'c', 'r', 'e', 't', 'k', 'e', 'y' };

std::vector<unsigned char> data = { 'T', 'h', 'i', 's', ' ', 'i', 's', ' ', 't', 'h', 'e', ' ', 'd', 'a', 't', 'a', ' ', 't', 'o', ' ', 'p', 'r', 'o', 't', 'e', 'c', 't', '.' };

// Initialize PDCPIntegrity with a key

PDCPIntegrity pdcp(key);

// Generate integrity protection tag

auto tag = pdcp.generate\_integrity\_protection\_tag(data);

std::cout << "Integrity Tag: " << bytes\_to\_hex(tag) << std::endl;

// Verify integrity

bool is\_valid = pdcp.verify\_integrity(data, tag);

std::cout << "Integrity Valid: " << (is\_valid ? "true" : "false") << std::endl;

return 0;

}