AuthQUIC

1. Service Description

For this project, I am planning to design and implement “AuthQUIC”, a stateful authentication protocol that built on top of the QUIC transport protocol. Its primary purpose is to authenticate clients to a server using a simple challenge-response authentication method. AuthQUIC is suitable for use cases like logging into chat or management systems, Device-to-server authentication, and session validation before encrypted file transfers.

The client begins the exchange by sending a HelloMessage that includes a unique client_id. In response, the server issues a ChallengeMessage containing a newly generated nonce. The client then concatenates its username, password, and the received nonce, computes a hash over this combination, and transmits it back in a ResponseMessage. Upon receipt, the server verifies the hash. If hash value matches the expected value, the server issues an AcceptMessage with a session token, otherwise it sends a RejectMessage. Any unexpected conditions or protocol violations are signaled by an ErrorMessage.

2. Message Definition (PDUs)

Key Constants:

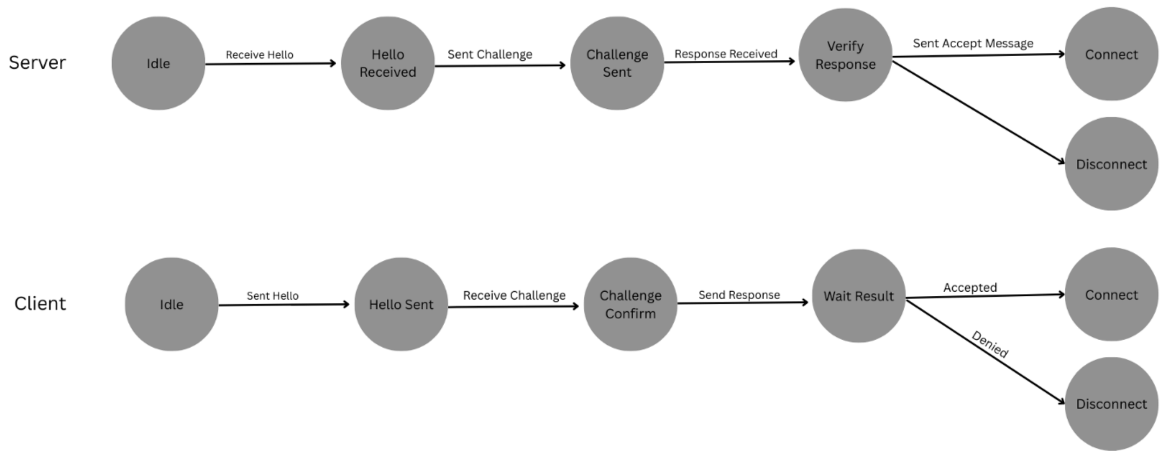
```
MSG_TYPE_HELLO      = 0x01
MSG_TYPE_CHALLENGE   = 0x02
MSG_TYPE_RESPONSE   = 0x03
MSG_TYPE_ACCEPT     = 0x04
MSG_TYPE_REJECT     = 0x05
MSG_TYPE_ERROR      = 0x06
PROTOCOL_VERSION    = 0x01
```

Header and Messages:

```
class PDUHeader:
  version: int    # 1 byte
  msg_type: int   # 1 byte
  flags: int      # 1 byte (can hold reserved bits or optional flags)
  length: int     # 2 bytes
```

```
class HelloMessage:
  header: PDUHeader
  client_id: str  # 1-byte length

class ChallengeMessage:
  header: PDUHeader
  nonce: bytes  # 16 bytes

class ResponseMessage:
  header: PDUHeader
  username: str      # 1-byte length
  response_hash: bytes  # 32 bytes (SHA256)

class AcceptMessage:
  header: PDUHeader
  session_token: bytes  # 16 bytes

class RejectMessage:
  header: PDUHeader
  reason: str  # 1-byte length

class ErrorMessage:
  header: PDUHeader
  error_code: int  # 2 bytes
  description: str # 1-byte length
```

3. DFA



4. Extensibility

The AuthQUIC protocol is designed with extensibility in mind to accommodate future enhancements without breaking compatibility. A version field in the message header allows clients and servers to negotiate compatible protocol versions and handle unsupported ones. To introduce new functionalities such as re-authentication, logout, session refresh, or token renewal, additional msg_type values can be defined. Reserved bits in the header are available for experimental features or future control flags. Furthermore, the protocol can support a capabilities message during the initial handshake to enable clients and servers to advertise and negotiate supported features.

5. Security Implications

The authentication mechanism relies on a challenge-response scheme using SHA-256. The server stores pre-hashed secrets, typically in the form of a hash of the username and password. To authenticate, the client proves knowledge of this secret by sending a hash of the username, password, and a server-provided nonce.

Confidentiality and integrity are ensured through QUIC's built-in use of TLS 1.3, which encrypts and authenticates all transmitted data. Replay protection is implemented by having the server generate a unique random nonce for each session, effectively preventing reuse of previous authentication attempts.

Additionally, access control can be enforced by maintaining an access control list (ACL) on the server, allowing it to verify user roles after successful authentication.