CHAP

CHallenge Authentication Protocol

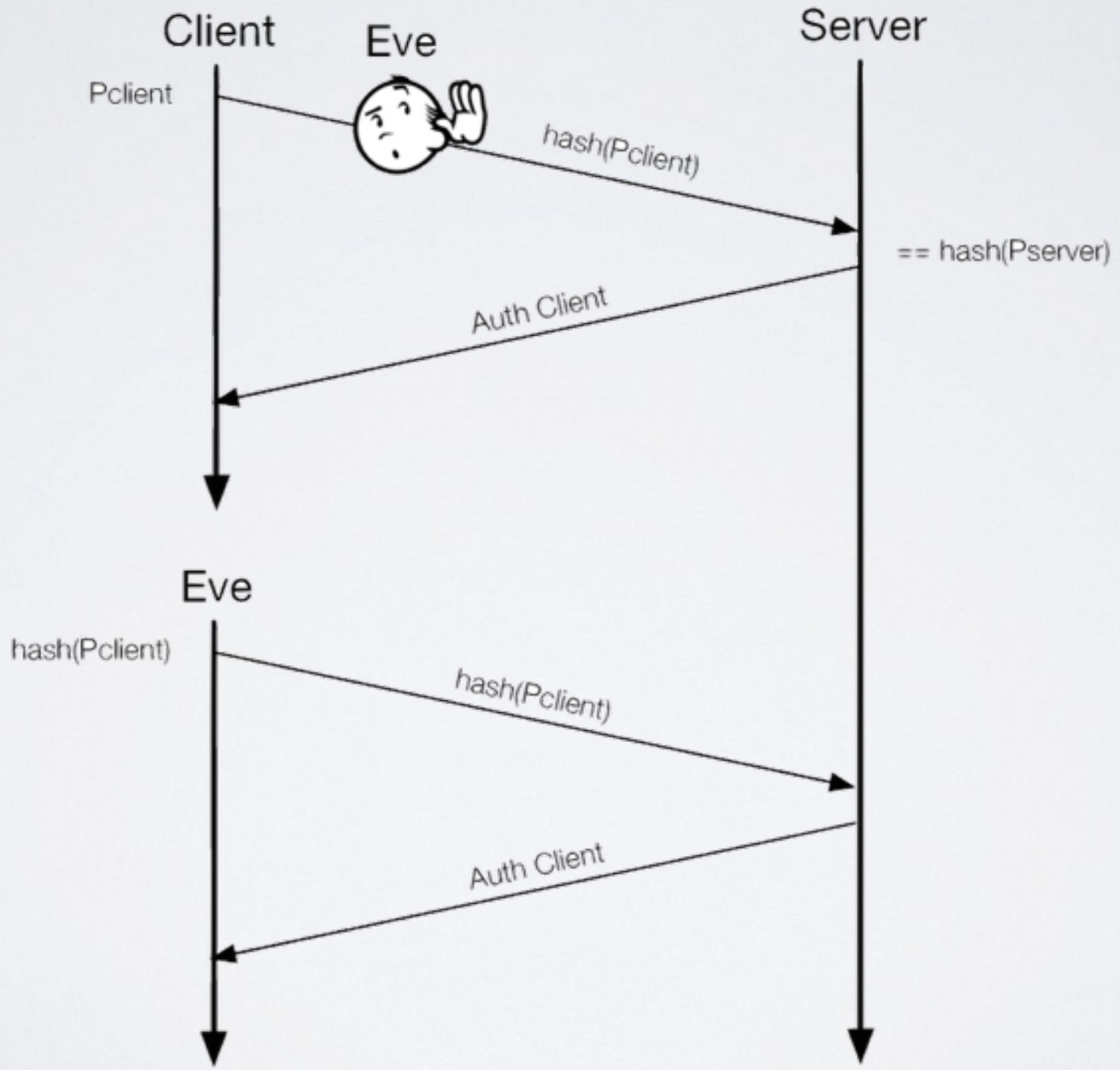
Paolo Bruzzo , Dario Casula
(pbruzz2@uic.edu , dcasul3@uic.edu)
CS 587 - Computer Systems Security - Fall 2014
University of Illinois at Chicago

WHAT IS IT ?

- **Authentication Protocol**
  - Prevents replay attacks
  - Works on non encrypted channels
  - Exploits the 3-way handshake logic
  - Keeps checking the authentication at run tim

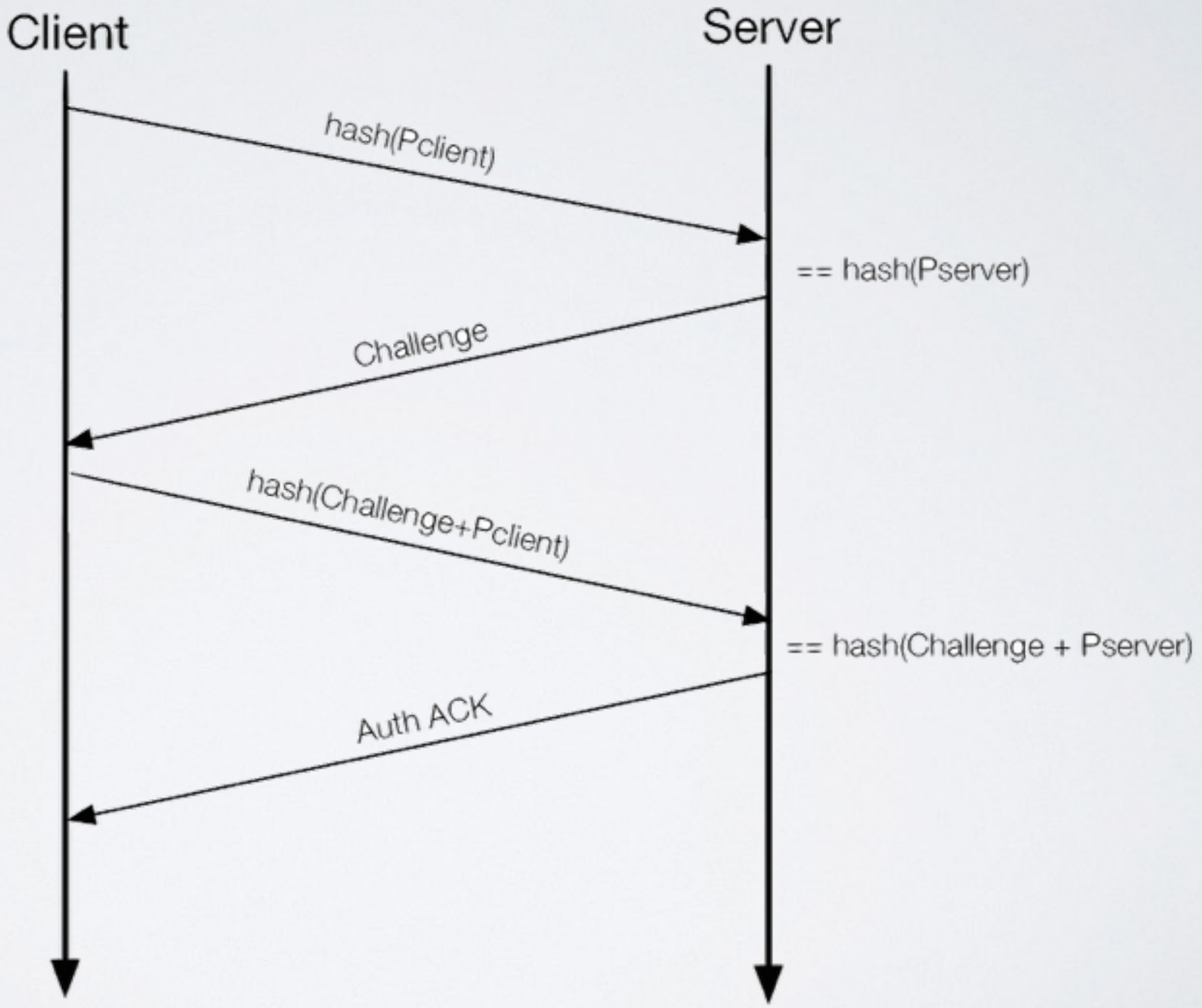
REPLAY ATTACK

- **Eve authenticates herself as Client**



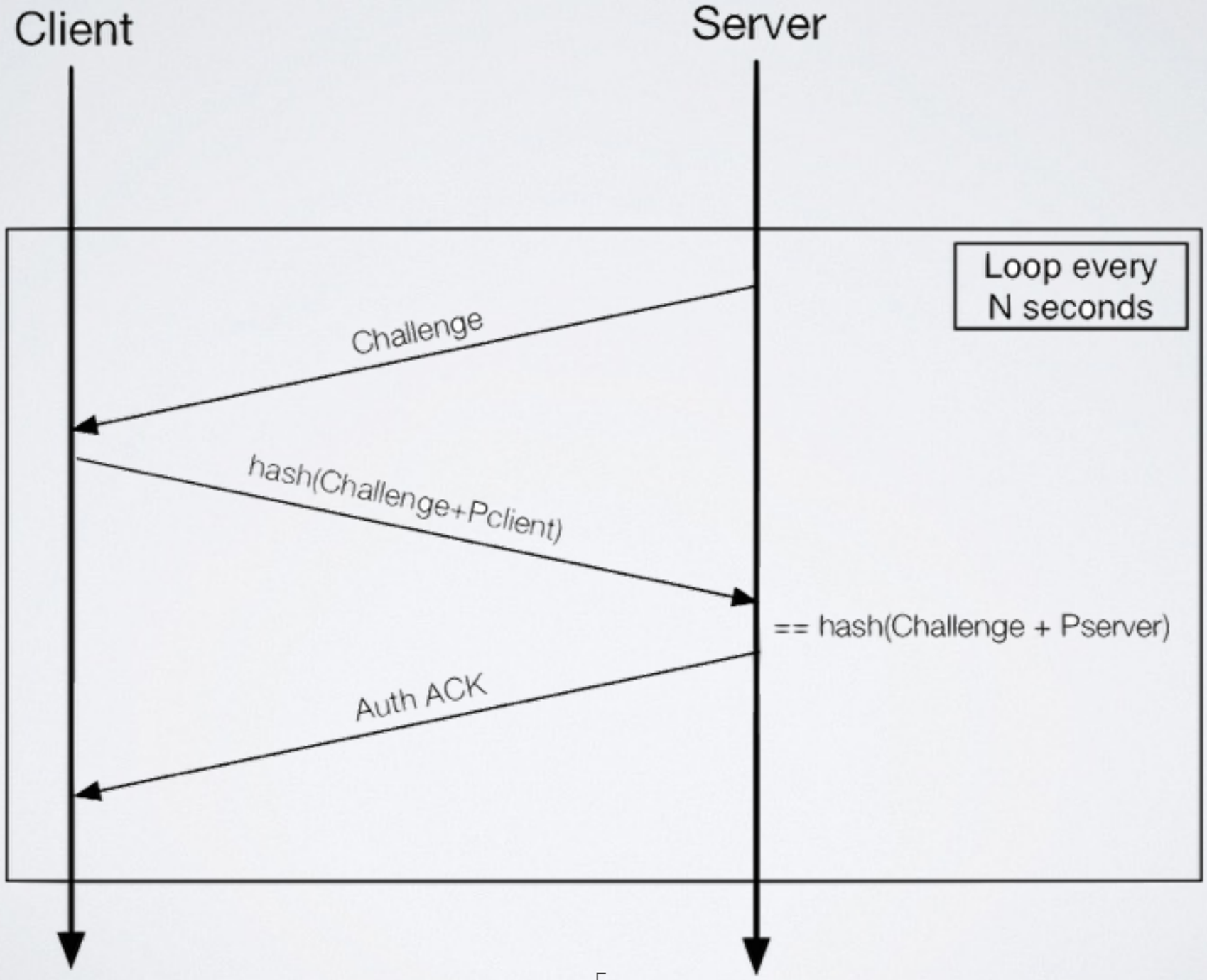
CHAP FIRST STEP

- **First step of CHAP**



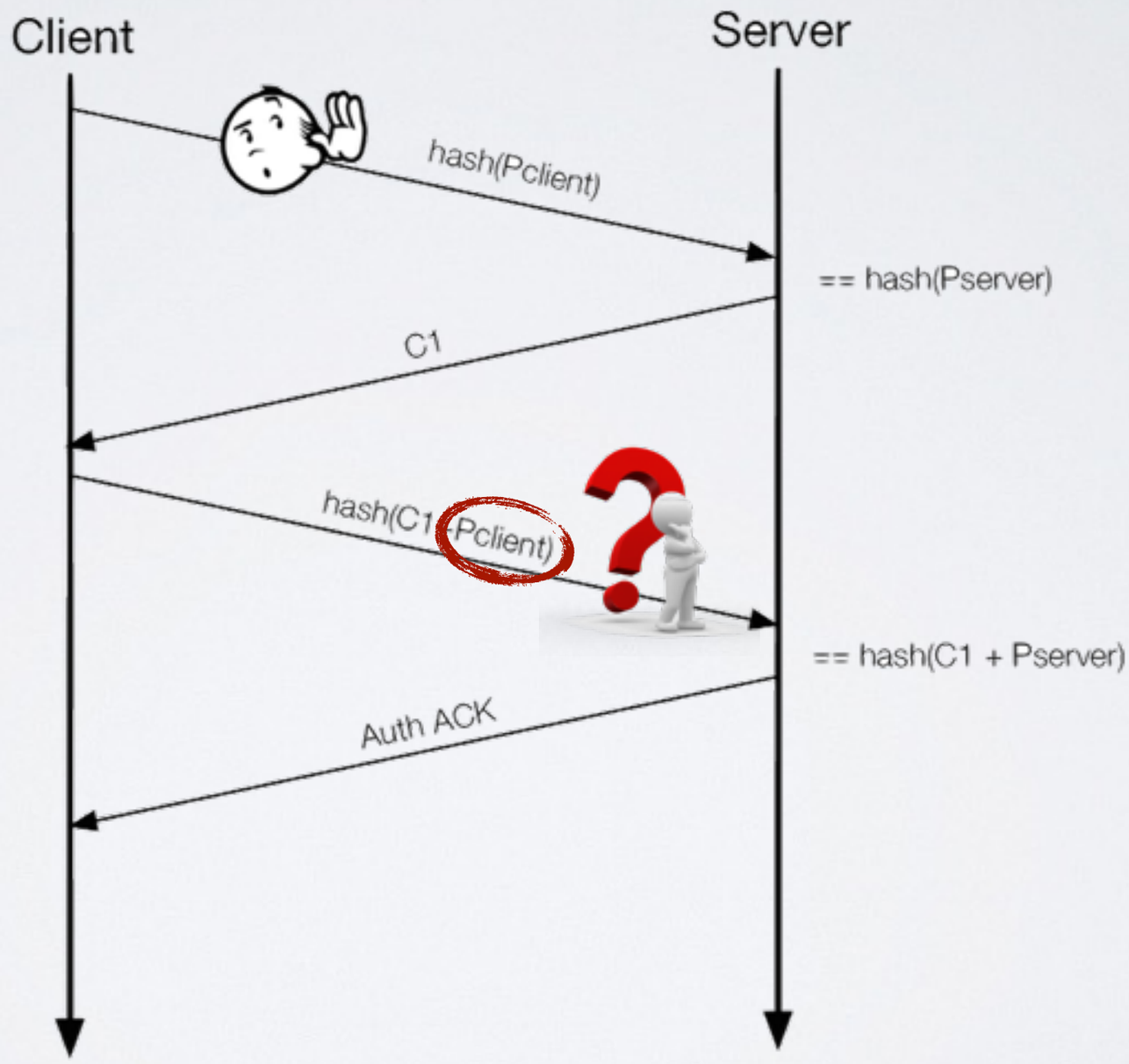
THE LOOP

- **Authentication keeps running after first step**



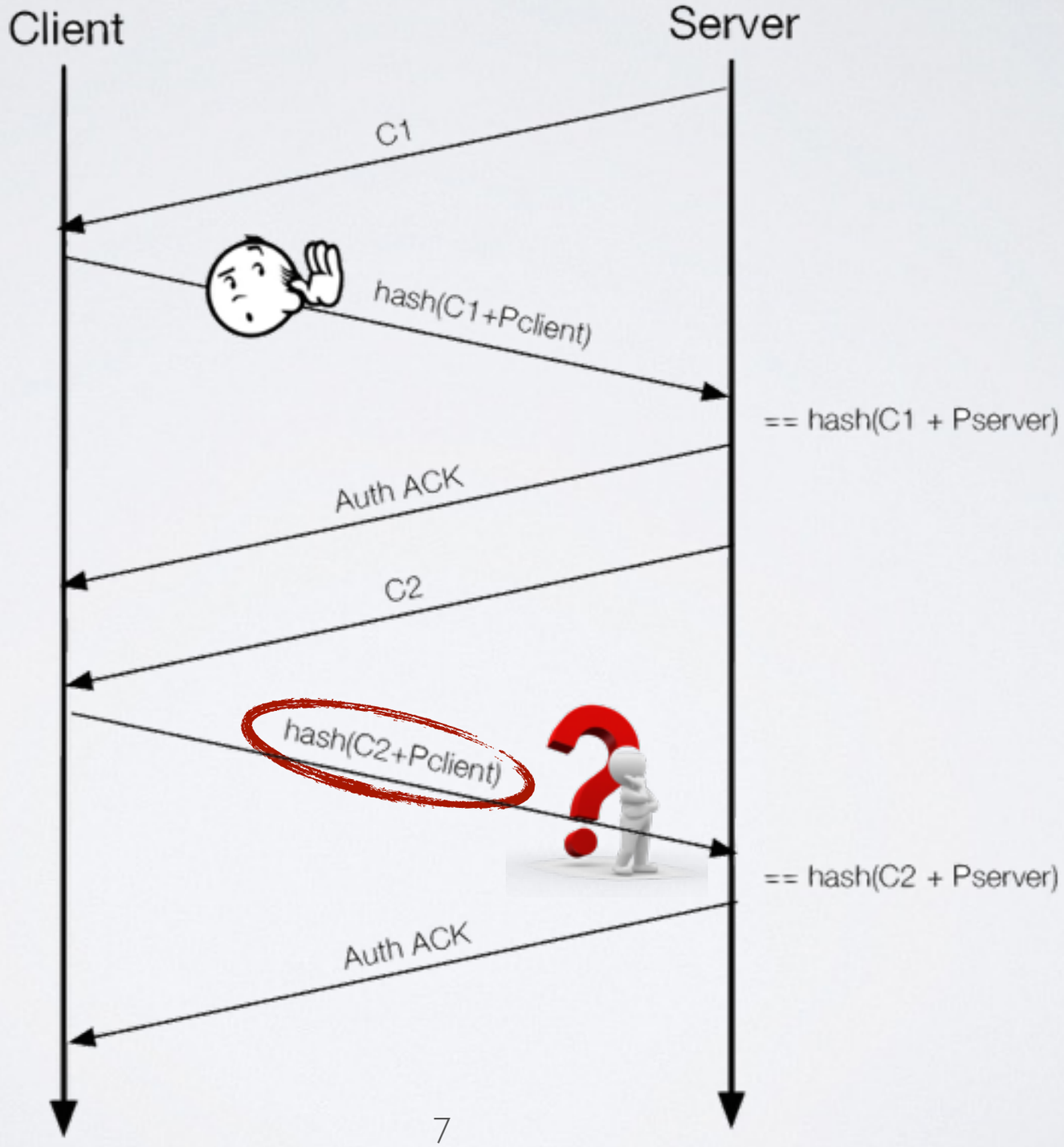
TRY TO ATTACK I

- **First attempt of attack by Eve**



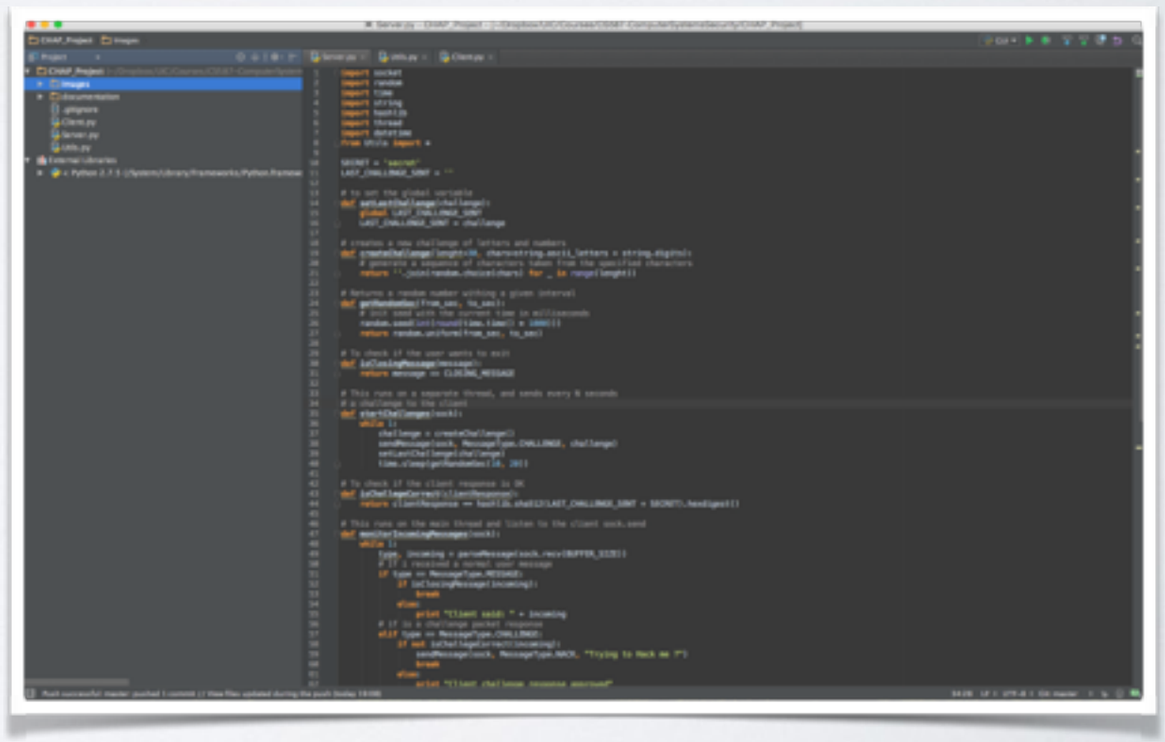
TRY TO ATTACK 2

- **Second attempt of attack by Eve**



OUR WORK

- **We implemented CHAP**
  - In Python
  - Focusing on the logic
  - Forgetting about the packets structure



CLIENT

python

- **Function to handle the challenges**
  - Runs on a separate thread

```
# Handles the challenge messages
def handleChallenges(sock, password):
    while 1:
        # Get the challenge
        type, challenge = parseMessage(sock.recv(BUFFER_SIZE))

        # Hash the challenge and the password together
        hashedChallenge = encode(challenge + password)

        # Send it back
        sendMessage(sock, MessageType.CHALLENGE, hashedChallenge)

        # Remember it
        setLastChallenge(challenge)

        # Receive back the response from the server (Accepted / Denied)
        type, response = parseMessage(sock.recv(BUFFER_SIZE))
        print response

        if type != MessageType.ACK:
            sock.close()
            os._exit(0)
```

SERVER

python

- **Functions to handle the challenges**
  - Send and Receive on separate threads

```
# This runs on a separate thread, and sends every N seconds
# a challenge to the client
def startChallenges(sock):
    while 1:
        challenge = createChallenge()
        sendMessage(sock, MessageType.CHALLENGE, challenge)
        setLastChallenge(challenge)
        time.sleep(getRandomSec(10, 20))

# This runs on the main thread and listen to the client
def monitorIncomingMessages(sock):
    while 1:
        type, incoming = parseMessage(sock.recv(BUFFER_SIZE))
        # If i received a normal user message
        if type == MessageType.MESSAGE:
            if isClosingMessage(incoming):
                break
            else:
                print "Client said: " + incoming
        # if is a challenge packet response
        elif type == MessageType.CHALLENGE:
            if not isChallegeCorrect(incoming):
                sendMessage(sock, MessageType.NACK, "Trying to Hack me ?")
                break
            else:
                sendMessage(sock, MessageType.ACK, 'Your authentication has been approved at')
```

LET'S PLAY A DEMO