Project Overview

**VANET Secure Routing Protocol Implementation in NS-2/NS-3 or OMNeT++**

**Project Overview:** Implement a digital signature and hash function-based approach for secure routing in Vehicular Ad Hoc Networks (VANET). The project aims to enhance the security of routing protocols in VANET by leveraging cryptographic techniques to prevent attacks like data tampering, impersonation, and message modification.

**Key Requirements:**

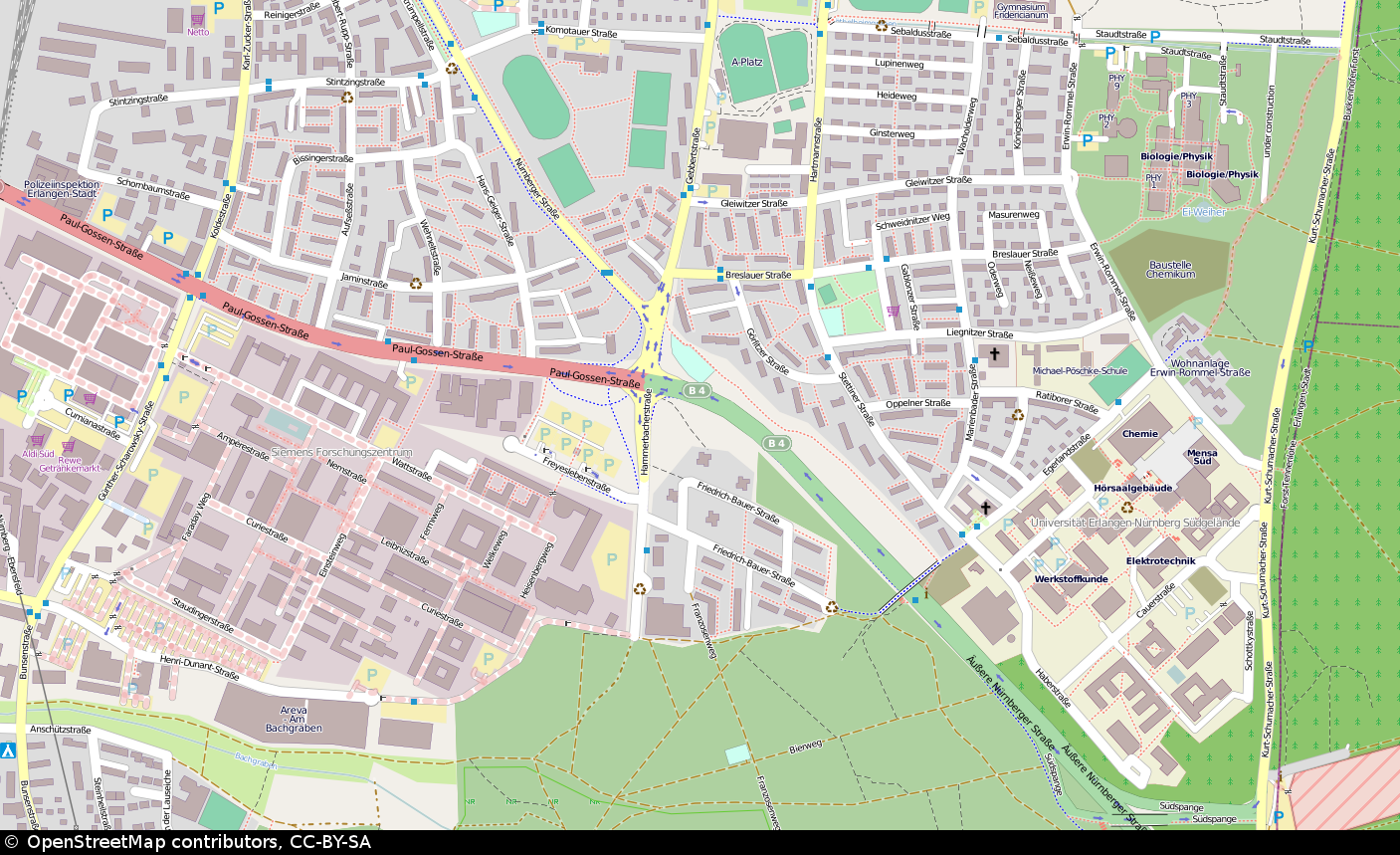
* **Simulation Platform:** Implementation in either NS-2, NS-3, or OMNeT++ with Veins and SUMO.
* **Cryptographic Techniques:** Digital signatures and hash functions to ensure message integrity and authentication.
* **Secure Routing Protocol:** Design and implementation of a secure, lightweight, and efficient routing protocol for VANET.
* **Attack Mitigation:** Simulation of common VANET-specific attacks and validation of the protocol’s ability to counter these threats.
* **Performance Metrics:** Evaluation of the protocol’s performance based on metrics like packet delivery ratio, end-to-end delay, throughput, and security effectiveness.

**Deliverables:**

* Complete source code for the implemented protocol.
* Configuration files and simulation setup.
* Documentation explaining the implementation, configuration steps, and usage.
* Detailed report on protocol performance, attack scenarios, and security analysis.

**Skills Required:**

* Strong expertise in network simulation tools (NS-2, NS-3, or OMNeT++ with Veins/SUMO).
* Proficiency in C++, Python, or related languages used in network simulation.
* Solid understanding of VANET architecture and protocols.
* Knowledge of cryptographic algorithms, especially digital signatures and hash functions.
* Experience in cybersecurity and secure routing protocol development is a plus.

For reference:  
import random

import hashlib

import matplotlib.pyplot as plt

import time

import pandas as pd # type: ignore

class Vehicle:

    def \_\_init\_\_(self, vehicle\_id, speed, position):

        self.id = vehicle\_id

        self.speed = speed

        self.position = position

        self.salt = "vanet" + str(random.random())  # Add a salt

    def move(self, dt):

        self.position = (self.position[0] + self.speed \* dt \* random.uniform(-0.1, 1.1),

                         self.position[1] + self.speed \* dt \* random.uniform(-0.1, 1.1))

    def check\_collision(self, other, threshold=1):

        distance = ((self.position[0] - other.position[0]) \*\* 2 +

                    (self.position[1] - other.position[1]) \*\* 2) \*\* 0.5

        return distance < threshold

    def generate\_message(self):

        message = {

            "vehicle\_id": self.id,

            "speed": self.speed,

            "position": self.position,

        }

        return message, self.hash\_message(message)

    def hash\_message(self, message):

        message\_bytes = str(message).encode()

        hashes = {}

        start\_time = time.time()

        hashes["sha256"] = hashlib.sha256(message\_bytes + self.salt.encode()).hexdigest()

        hashes["sha256\_time"] = time.time() - start\_time

        start\_time = time.time()

        hashes["md5"] = hashlib.md5(message\_bytes + self.salt.encode()).hexdigest()

        hashes["md5\_time"] = time.time() - start\_time

        start\_time = time.time()

        hashes["sha1"] = hashlib.sha1(message\_bytes + self.salt.encode()).hexdigest()

        hashes["sha1\_time"] = time.time() - start\_time

        start\_time = time.time()

        hashes["blake2b"] = hashlib.blake2b(message\_bytes + self.salt.encode()).hexdigest()

        hashes["blake2b\_time"] = time.time() - start\_time

        start\_time = time.time()

        hashes["sha3\_256"] = hashlib.sha3\_256(message\_bytes + self.salt.encode()).hexdigest()

        hashes["sha3\_256\_time"] = time.time() - start\_time

        return hashes

    def check\_integrity(self, message, hashes):

        for hash\_type, hash\_value in hashes.items():

            if hash\_type in ["sha256", "md5", "sha1", "blake2b", "sha3\_256"]:

                if hash\_value!= self.hash\_message(message)[hash\_type]:

                    return False

        return True

    def receive\_message(self, message, hashes):

        print(f"Vehicle {self.id} received message from {message['vehicle\_id']}: {message}")

        print(f"Received hashes: {hashes}")

        if self.check\_integrity(message, hashes):

            print("Message integrity is valid.")

        else:

            print("Message integrity is NOT valid!")

def simulate(vehicles, dt, num\_steps):

    hash\_times = {"sha256": [], "md5": [], "sha1": [], "blake2b": [], "sha3\_256": []}

    for \_ in range(num\_steps):

        for vehicle in vehicles:

            vehicle.move(dt)

            collisions = [other for other in vehicles if vehicle!= other and vehicle.check\_collision(other)]

            if collisions:

                print(f"Collision detected between vehicle {vehicle.id} and: {', '.join(other.id for other in collisions)}")

            message, hashes = vehicle.generate\_message()

            for other in vehicles:

                if hasattr(other, 'receive\_message'):

                    other.receive\_message(message.copy(), hashes.copy())

            for hash\_type, hash\_time in hashes.items():

                if hash\_type.endswith("\_time"):

                    hash\_times[hash\_type[:-5]].append(hash\_time)

    # Prepare data for Pandas boxplot

    hash\_data = []

    for hash\_type, times in hash\_times.items():

        for time in times:

            hash\_data.append({"hash\_type": hash\_type, "time": time})

    # Convert hash\_data to a DataFrame

    hash\_df = pd.DataFrame(hash\_data)

    # Create a boxplot using Pandas

    hash\_df.boxplot(column="time", by="hash\_type", showmeans=True)

    plt.title("Hash Generation Times")

    plt.ylabel("Time (s)")

    plt.xlabel("Hash Function")

    plt.xticks(rotation=45)  # Rotate hash type labels for better readability

    plt.show()

def plot\_speeds(vehicles, num\_steps, dt):

    times = list(range(num\_steps))

    speeds = {vehicle.id: [] for vehicle in vehicles}

    for i in range(num\_steps):

        for vehicle in vehicles:

            vehicle.move(dt)

            speeds[vehicle.id].append(vehicle.speed)

        collisions = [other for other in vehicles if vehicle!= other and vehicle.check\_collision(other)]

        if collisions:

            print(f"Collision detected between vehicle {vehicle.id} and: {', '.join(other.id for other in collisions)}")

    # Create a pandas DataFrame

    speeds\_df = pd.DataFrame(speeds, index=times)

    # Plot speeds using matplotlib

    speeds\_df.plot(figsize=(10, 5), title="Vehicle Speeds Over Time", legend=True)

    plt.xlabel("Time")

    plt.ylabel("Speed")

    plt.show()

def plot\_positions(vehicles, num\_steps, dt):

    positions = {vehicle.id: [] for vehicle in vehicles}

    for i in range(num\_steps):

        for vehicle in vehicles:

            vehicle.move(dt)

            positions[vehicle.id].append(vehicle.position)

        collisions = [other for other in vehicles if vehicle!= other and vehicle.check\_collision(other)]

        if collisions:

            print(f"Collision detected between vehicle {vehicle.id} and: {', '.join(other.id for other in collisions)}")

    # Create a pandas DataFrame

    positions\_df = pd.DataFrame(positions)

    # Plot positions using matplotlib

    positions\_df.apply(pd.Series.explode).plot.scatter(x=0, y=1, figsize=(10, 5), title="Vehicle Positions Over Time", legend=True)

    plt.xlabel("X Position")

    plt.ylabel("Y Position")

    plt.show()

vehicle1 = Vehicle("V1", 65, (0, 0))

vehicle2 = Vehicle("V2", 50, (10, 20))

vehicle3 = Vehicle("V3", 35, (30, 50))

vehicle4 = Vehicle("V4", 20, (10, 50))

vehicle5 = Vehicle("V5", 95, (30, 10))

vehicle6 = Vehicle("V6", 70, (70, 10))

# Valid message

message, hashes = vehicle1.generate\_message()

vehicle2.receive\_message(message, hashes.copy())

# Invalid message (tampered speed)

tampered\_message = message.copy()

tampered\_message["speed"] = 100

vehicle2.receive\_message(tampered\_message, hashes.copy())

# Valid message

message, hashes = vehicle3.generate\_message()

vehicle4.receive\_message(message, hashes.copy())

# Invalid message (tampered speed)

tampered\_message = message.copy()

tampered\_message["speed"] = 100

vehicle4.receive\_message(tampered\_message, hashes.copy())

# Simulate hash generation times

simulate([vehicle1, vehicle2], 0.1, 1000)

# Plot speeds

plot\_speeds([vehicle1, vehicle2], 1000, 2000)

# Plot positions

plot\_positions([vehicle1, vehicle2], 1000, 0.1)

# Simulate hash generation times

simulate([vehicle3, vehicle4], 0.1, 1000)

# Plot speeds

plot\_speeds([vehicle3, vehicle4], 1000, 2000)

# Plot positions

plot\_positions([vehicle3, vehicle4], 1000, 0.1)