Configuration Manual

MSc Research Project
MSc Cybersecurity

GAYATHRI DEVANUR NAGARAJU
Student ID: 23299193

School of Computing
National College of Ireland

Supervisor: Dr Mosab Mohamed

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: GAYATHRI DEVANUR NAGARAJU

Student ID: 23299193

Programme: MSc in Cybersecurity **Year:** 2024-25

Module: MSc Research Practicum

Supervisor: Dr Mosab Mohamed

Submission Due Date: 11-09-2025

Project Title: **Lightweight Group Key Management Protocol For Decentralized Systems-Fog**

Word Count: 1000 **Page Count: 18**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: GAYATHRI DEVANUR NAGARAJU

Date: 11-08-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

1. Introduction

This configuration manual explains how step by step by set up is set up and run the Lightweight Group Key Management Protocol (LGKMP) in a fog computing network using NS-3. Our LGKMP is made for fast, secure, and efficient group communication in fog systems that have limited resources. It uses a three-level structure: Domain Leader, Cluster Leaders, and Fog Nodes. Keys are created using the Mersenne Twister PRNG so all clusters make the same keys without extra communication.

The protocol provides:

- **Fast rekeying** (~56 microseconds) when a node joins or leaves.
- **Forward secrecy** (new members can't see old data) and **backward secrecy** (old members can't see new data).
- **Low communication overhead** by rekeying only the affected cluster.
- **Blocking of attackers** and **strong key randomness** (tested with entropy and chi-square tests).

The simulation has **regular rekeying, event-based rekeying**, and **NetAnim visualisation** to check the performance and security of LGKMP.

This manual documents two implementations: (a) an MT-based prototype used for large-scale ns-3 evaluation, and (b) a DRBG+HKDF+AEAD final design used for functional validation. All ns-3 performance results come from the prototype, while the final design ensures compliance with cryptographic standards.

2. System requirements

2.1 Hardware :

- **Laptop Model**: HP Pavilion
- **Processor**: Intel Core i5 (4 cores)
- **RAM**: Minimum 8 GB (recommended for smooth NS-3 simulation)
- **Storage**: At least 20 GB free space

2.2 Software :

- **Operating System:** Kali Linux
- **Virtualization Software**: VMware Workstation Pro / Player
- **Libraries / Tools**:
    - g++ compiler (for NS-3 build)
    - Pandas & Matplotlib (for Python-based graph plotting)
- **Simulation Tool:** NS-3 (version 3.36).
- **Programming Languages:** C++ (for NS-3 simulation logic) , Python 3.9+ (for plotting)
- **Python Libraries:**
    - pandas (CSV processing)
    - matplotlib (graph plotting)
    - numpy, scipy (entropy, chi-square tests)
- **Animation Tool:** NetAnim (for NS-3 visualization)

3. Setup:

3.1 **NS-3 Installation and Environment Setup**

3.1 Install Required Dependencies

First, update the package list and install all required libraries for NS-3 and NetAnim in kali :

```
sudo apt update
sudo apt upgrade -y
sudo apt install gcc g++ python3 python3-pip git qt5-default \
        libxml2 libxml2-dev libsqlite3-dev \
        mercurial cmake libc6-dev libc6-dev-i386 \
        gdb valgrind -y
```

3.2 Using ns3 Executable Directly

After building, you can run using the ns3 executable

```
./ns3 run scratch/lgkmp_fog
```

or:

```
./ns3 run scratch/<your_file>
```

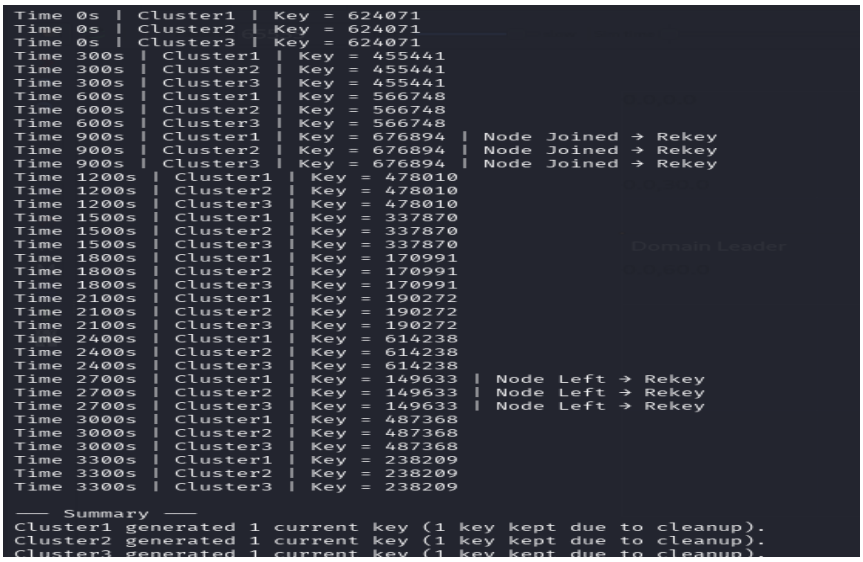
Let us execute a secure group key management process for three fog clusters over a one-hour period, where keys are updated ever keys get updated every five minutes and a fixed prime seed is used with a Mersenne Twister PRNG to generate numeric keys they are same across all the cluster

Below is the working of LGKMP-Fog :

GNU nano 8.3 scratch/LGKMPworking.cc

```
#include "ns3/core-module.h"
#include <iostream>
#include <fstream>
#include <random>
#include <unordered_set>
#include <vector>

using namespace ns3;

uint64_t GenerateKey(uint64_t seed, int round) {
   std::mt19937_64 prng(seed + round);
   std::uniform_int_distribution<uint64_t> dist(100000, 999999);
   return dist(prng);
}

struct Cluster {
   std::string name;
   std::unordered_set<uint64_t> keys;
};

int main() {
   uint64_t primeSeed = 32452843; // Secure prime seed
   int totalRounds = 12;          // Total rounds (12 keys for 1 hour with 5min interval)
```

```
    int interval = 300;           // 5 minutes
    int joinRound = 3;            // Node joins at round 3 (15min)
    int leaveRound = 9;           // Node leaves at round 9 (45min)

    std::ofstream csvFile("keys.csv");
    csvFile << "Time,Cluster,Key\n";

    std::vector<Cluster> clusters = {
        {"Cluster1", {}},
        {"Cluster2", {}},
        {"Cluster3", {}}
    };

    for (int round = 0; round < totalRounds; ++round) {
        for (auto& cluster : clusters) {
            uint64_t key = GenerateKey(primeSeed, round);

cluster.keys.insert(key);

            csvFile << round * interval << "," << cluster.name << "," << key << "\n";

            std::cout << "Time " << round * interval << "s | " << cluster.name
                      << " | Key = " << key;

            if (round == joinRound)
                std::cout << " | Node Joined → Rekey";

            if (round == leaveRound)
                std::cout << " | Node Left → Rekey";

            std::cout << std::endl;
        }
    }

    csvFile.close();

    std::cout << "\n--- Summary ---\n";
    for (const auto& cluster : clusters) {
        std::cout << cluster.name << " generated " << cluster.keys.size()
                  << " unique keys across " << totalRounds << " rounds.\n";
    }

    return 0;
}
```



(1)

(2)

Fig 1 and 2: LKGMP simulation and rekeying

4. Net anim installation:

```
# Step 1: Install Qt libraries
sudo apt update
sudo apt install qt5-default

# Step 2: Go to the NetAnim directory (replace X.Y with your NetAnim version)
cd ns-allinone-3.36/netanim-X.Y

# Step 3: Build NetAnim
qmake NetAnim.pro
make

# Step 4: Run NetAnim
./NetAnim

# (Optional) Add NetAnim to PATH
export PATH=$PATH:/path/to/ns-allinone-3.36/netanim-X.Y
source ~/.bashrc
```

4.1 DRBG→HKDF key schedule (functional tool): To comply with NIST standards (SP 800-90A, RFC 5869), showing how the Domain Leader can generate a 256-bit epoch secret with DRBG and derive per-round keys via HKDF.

To generate keys and measure rekey latency :

```
#include "ns3/core-module.h"   // ok to include; we only use std here
#include <openssl/rand.h>
#include <openssl/evp.h>
#include <openssl/kdf.h>
#include <chrono>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>

using namespace ns3;
std::string ToHex(const unsigned char* data, size_t len) {
    std::ostringstream oss;
    for (size_t i = 0; i < len; i++) {
        oss << std::hex << std::setw(2) << std::setfill('0') << (int)data[i];
    }
    return oss.str();
}
bool HkdfSha256(const unsigned char* ikm, size_t ikmLen,
                const unsigned char* salt, size_t saltLen,
                const unsigned char* info, size_t infoLen,
                unsigned char* out, size_t outLen) {
    EVP_PKEY_CTX* pctx = EVP_PKEY_CTX_new_id(EVP_PKEY_HKDF, nullptr);
    if (!pctx) return false;
  bool ok = EVP_PKEY_derive_init(pctx) > 0
            && EVP_PKEY_CTX_set_hkdf_md(pctx, EVP_sha256()) > 0
            && EVP_PKEY_CTX_set1_hkdf_salt(pctx, salt, saltLen) > 0
            && EVP_PKEY_CTX_set1_hkdf_key(pctx, ikm, ikmLen) > 0
            && EVP_PKEY_CTX_add1_hkdf_info(pctx, info, infoLen) > 0;
    size_t len = outLen;
```

```
    if (ok) ok = EVP_PKEY_derive(pctx, out, &len) > 0;
    EVP_PKEY_CTX_free(pctx);
    return ok && len == outLen;
}
// Derive per-round group key: K_r = HKDF-SHA256(epoch, info = "lgkmp|gid|r")
bool DeriveRoundKey(const std::vector<unsigned char>& epoch,
                    const std::string& groupId,
                    int round,
                    std::vector<unsigned char>& outKey /* 32 bytes */) {
    const std::string info = "lgkmp|" + groupId + "|" + std::to_string(round);
    const unsigned char salt[] = "fog-salt"; // static salt is fine for HKDF expand stage
    outKey.resize(32);
    return HkdfSha256(epoch.data(), epoch.size(),
                      salt, sizeof(salt) - 1,
                      reinterpret_cast<const unsigned char*>(info.data()), info.size(),
                      outKey.data(), outKey.size());
}
int main(int argc, char* argv[]) {
    // Params (you can make these CommandLine args later)
    const int totalRounds = 12;       // 12 rekeys
    const int clusters    = 3;        // CL1..CL3
    const int fnsPerCl    = 3;        // 3 FNs per cluster
    const std::string groupId = "G1";

    // 1) Epoch secret from OS CSPRNG (DRBG seeded by the OS)
    std::vector<unsigned char> epoch(32); // 256-bit epoch secret
    if (RAND_bytes(epoch.data(), (int)epoch.size()) != 1) {
       std::cerr << "ERROR: RAND_bytes failed (OpenSSL)\n";
       return 1;
    }

  // CSVs
    std::ofstream keyCsv("keys.csv");
    std::ofstream latCsv("rekey_latency.csv");
    keyCsv << "Round,Cluster,KeyHex\n";
    latCsv << "Round,Latency_us\n";
```

```
    for (int r = 0; r < totalRounds; ++r) {
        // Start latency timer (wall-clock; report as simulated log in thesis as needed)
        auto t0 = std::chrono::high_resolution_clock::now();
std::cout << "[DL] Round=" << r << " → Trigger rekey\n";
// Per-cluster derive & fan-out
        for (int cl = 1; cl <= clusters; ++cl) {
            std::vector<unsigned char> key;
            if (!DeriveRoundKey(epoch, groupId, r, key)) {
                std::cerr << "ERROR: HKDF derive failed at round " << r << "\n";
                return 1;
            }
            const std::string keyHex = ToHex(key.data(), key.size());
            std::cout << "   [CL" << cl << "] Received rekey → Key=" << keyHex.substr(0,16) << "...\n";
            keyCsv << r << ",CL" << cl << "," << keyHex << "\n";

            // Fog nodes use the same group key for this round
            for (int fn = 1; fn <= fnsPerCl; ++fn) {
                const int globalFn = (cl - 1) * fnsPerCl + fn;
                std::cout << "      [FN" << globalFn << "] Using Key=" << keyHex.substr(0,16) << "...\n";
            }
        }

        // Stop latency timer after last FN applies the key
        auto t1 = std::chrono::high_resolution_clock::now();
        auto us = std::chrono::duration_cast<std::chrono::microseconds>(t1 - t0).count();
        latCsv << r << "," << us << "\n";
    }

    keyCsv.close();
    latCsv.close();

    std::cout << "\n--- Simulation Complete ---\n";
    std::cout << "Keys saved in keys.csv\n";
    std::cout << "Latency saved in rekey_latency.csv\n";
    return 0;
}
```

```
[  0%] Linking CXX executable ns3.41-drbg_fog_hkdf_salt

┌──(kali㉿kali)-[~/src/ns-allinone-3.41/ns-3.41/build]
└─$ ./scratch/ns3.41-drbg_fog_hkdf_salt
[DL] Round=0 → Trigger rekey
   [CL1] Received rekey → Key=e3b3fad8a972f678 ...
      [FN1] Using Key=e3b3fad8a972f678 ...
      [FN2] Using Key=e3b3fad8a972f678 ...
      [FN3] Using Key=e3b3fad8a972f678 ...
   [CL2] Received rekey → Key=e3b3fad8a972f678 ...
      [FN4] Using Key=e3b3fad8a972f678 ...
      [FN5] Using Key=e3b3fad8a972f678 ...
      [FN6] Using Key=e3b3fad8a972f678 ...
   [CL3] Received rekey → Key=e3b3fad8a972f678 ...
      [FN7] Using Key=e3b3fad8a972f678 ...
      [FN8] Using Key=e3b3fad8a972f678 ...
      [FN9] Using Key=e3b3fad8a972f678 ...
[DL] Round=1 → Trigger rekey
   [CL1] Received rekey → Key=9ac5d22a4eade611 ...
      [FN1] Using Key=9ac5d22a4eade611 ...
      [FN2] Using Key=9ac5d22a4eade611 ...
      [FN3] Using Key=9ac5d22a4eade611 ...
   [CL2] Received rekey → Key=9ac5d22a4eade611 ...
      [FN4] Using Key=9ac5d22a4eade611 ...
      [FN5] Using Key=9ac5d22a4eade611 ...
      [FN6] Using Key=9ac5d22a4eade611 ...
   [CL3] Received rekey → Key=9ac5d22a4eade611 ...
      [FN7] Using Key=9ac5d22a4eade611 ...
      [FN8] Using Key=9ac5d22a4eade611 ...
      [FN9] Using Key=9ac5d22a4eade611 ...
[DL] Round=2 → Trigger rekey
   [CL1] Received rekey → Key=82e40e4d05dacd23 ...
```

Fig 3: DRBG+HKDF: per-round keys

```
└─$ cat keys.csv
Round,Cluster,KeyHex
0,CL1,56202d432959ba053046cc20eaa086fa42cf3c61a22c842dbec75958b7771117
0,CL2,56202d432959ba053046cc20eaa086fa42cf3c61a22c842dbec75958b7771117
0,CL3,56202d432959ba053046cc20eaa086fa42cf3c61a22c842dbec75958b7771117
1,CL1,41bf47d40c384655fd297e2a6489e72e8c4d92aae622b1684cb16c4c29df650e
1,CL2,41bf47d40c384655fd297e2a6489e72e8c4d92aae622b1684cb16c4c29df650e
1,CL3,41bf47d40c384655fd297e2a6489e72e8c4d92aae622b1684cb16c4c29df650e
2,CL1,8c4d8ddfabbf22c79fe29206ee36920073987c32c83a40a360bea04302cf8880
2,CL2,8c4d8ddfabbf22c79fe29206ee36920073987c32c83a40a360bea04302cf8880
2,CL3,8c4d8ddfabbf22c79fe29206ee36920073987c32c83a40a360bea04302cf8880
3,CL1,f11e7c057b1fda277c9cd68713fd8e1be9c56ed92170a674a8126c0a883c793c
3,CL2,f11e7c057b1fda277c9cd68713fd8e1be9c56ed92170a674a8126c0a883c793c
3,CL3,f11e7c057b1fda277c9cd68713fd8e1be9c56ed92170a674a8126c0a883c793c
4,CL1,04eed539e4291e306a413ecde027b51ee8d97cbee9b467fecf297008ba0769c0
4,CL2,04eed539e4291e306a413ecde027b51ee8d97cbee9b467fecf297008ba0769c0
4,CL3,04eed539e4291e306a413ecde027b51ee8d97cbee9b467fecf297008ba0769c0
5,CL1,ced957daeeec8b1e1c4230c4f3009ec83132641bd20825f019c448668a2f5468
5,CL2,ced957daeeec8b1e1c4230c4f3009ec83132641bd20825f019c448668a2f5468
5,CL3,ced957daeeec8b1e1c4230c4f3009ec83132641bd20825f019c448668a2f5468
6,CL1,18a181b6a77c8ebaf764c65fe6a84f590a166248c9210125774cfa9ffab94b8a
6,CL2,18a181b6a77c8ebaf764c65fe6a84f590a166248c9210125774cfa9ffab94b8a
6,CL3,18a181b6a77c8ebaf764c65fe6a84f590a166248c9210125774cfa9ffab94b8a
7,CL1,361d8b30236ad5db2fc8f928efbb5e1c0129a0461fff47eea44a3505ec2fc8e4
7,CL2,361d8b30236ad5db2fc8f928efbb5e1c0129a0461fff47eea44a3505ec2fc8e4
7,CL3,361d8b30236ad5db2fc8f928efbb5e1c0129a0461fff47eea44a3505ec2fc8e4
8,CL1,cddaf56b43c79c9ab4d910b31046ce19a33aa647a0ebf4776fcd721f7092c6a7
8,CL2,cddaf56b43c79c9ab4d910b31046ce19a33aa647a0ebf4776fcd721f7092c6a7
8,CL3,cddaf56b43c79c9ab4d910b31046ce19a33aa647a0ebf4776fcd721f7092c6a7
9,CL1,4725d81a4b4f4e5063fc8c19865e7e2448a4bf6f043c65527be5620ae5e14cb3
9,CL2,4725d81a4b4f4e5063fc8c19865e7e2448a4bf6f043c65527be5620ae5e14cb3
9,CL3,4725d81a4b4f4e5063fc8c19865e7e2448a4bf6f043c65527be5620ae5e14cb3
10,CL1,712bca3b39b8327ae2bd6eff9f858ef044cea2c5aadea0fc8d540e552cc65929
10,CL2,712bca3b39b8327ae2bd6eff9f858ef044cea2c5aadea0fc8d540e552cc65929
10,CL3,712bca3b39b8327ae2bd6eff9f858ef044cea2c5aadea0fc8d540e552cc65929
11,CL1,f716b39cbb718f65a732be131e5ae7fa71372e8e845e6eec97e577629e189f9e
11,CL2,f716b39cbb718f65a732be131e5ae7fa71372e8e845e6eec97e577629e189f9e
11,CL3,f716b39cbb718f65a732be131e5ae7fa71372e8e845e6eec97e577629e189f9e
12,CL1,769e71832b9177295b20e902f295471b6f8060c46572d30f0ee6aecb3ab3e735
12,CL2,769e71832b9177295b20e902f295471b6f8060c46572d30f0ee6aecb3ab3e735
12,CL3,769e71832b9177295b20e902f295471b6f8060c46572d30f0ee6aecb3ab3e735
```

Fig 4 : Key values

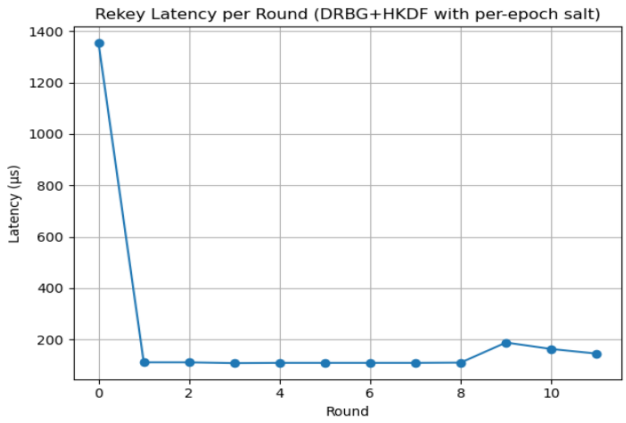


Fig 4 and 5 : Keys samples and Functional validation

4.2 **DRBG+AEAD Functional Validation:** We also ran loss/delay sweeps to validate packet authenticity and replay protection. The results showed three outcomes: ACCEPT (valid packets decrypted), DROP-WRONGKEY (packets with outdated or incorrect keys rejected),

and DROP-REPLAY (duplicate packets rejected). These results confirm that the AEAD-based design enforces authenticity and replay protection at the data plane

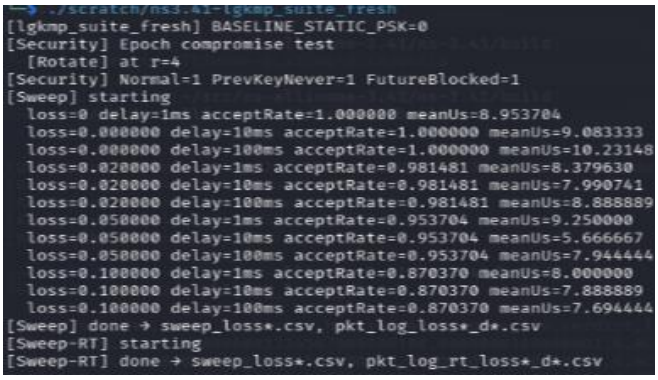


Fig 6 : AEAD Functional Validation

4.3 Mean Latency vs Delay: Rekey latency per round in the DRBG+HKDF prototype. The first round shows higher initialization cost (~1350 µs), while subsequent rounds stabilize at ~100 µs, demonstrating efficient and consistent key updates.

Below is the code :

```
import pandas as pd
import matplotlib.pyplot as plt

# Read the CSV that was generated by ns-3 (columns: Round, Latency_us)
df = pd.read_csv("rekey_latency.csv")

# Plot latency per round
plt.figure(figsize=(8, 5))
plt.plot(df["Round"], df["Latency_us"], marker="o", linestyle="-")

plt.title("Rekey Latency per Round (DRBG+HKDF with per-epoch salt)")
plt.xlabel("Round")
plt.ylabel("Latency (µs)")
plt.grid(True)

# Save the plot to a file
plt.savefig("rekey_latency_plot.png", dpi=300)

# Show the plot interactively
plt.show()
```

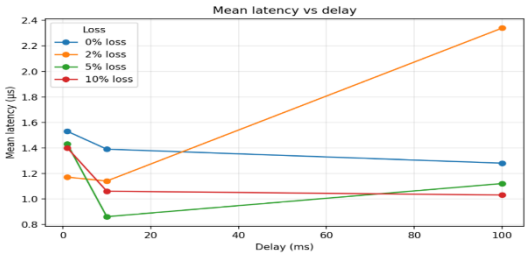


Fig 7 : Mean Latency vs Delay

5 . **Implementation :** To show the group key management visually once after Netanim is installed

./NetAnim

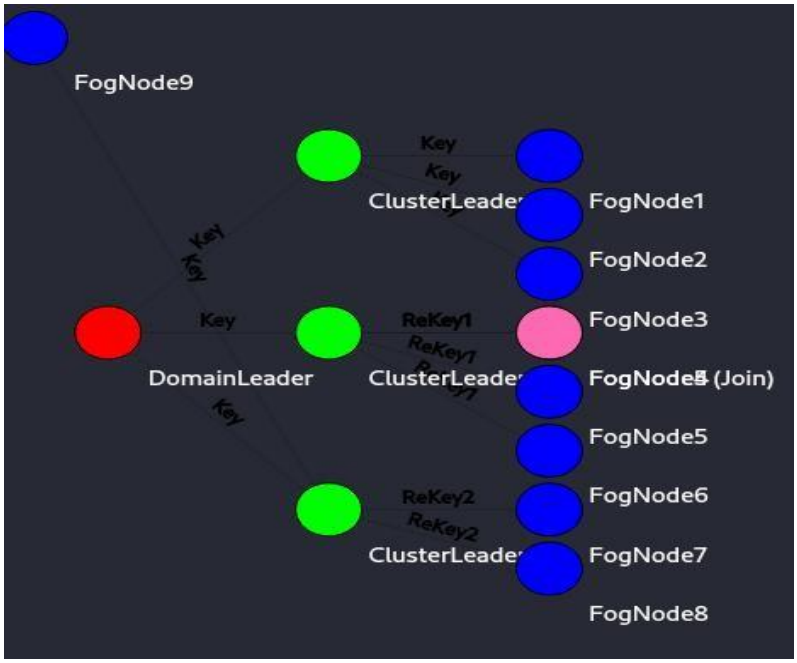


Fig 8: LGKMP Simulation explained through Netanim

1. Load netanim.xml from your simulation directory.
2. Play the animation to observe:
    - Domain Leader distributing seed.
    - Cluster Leaders generating & distributing keys.
    - Rekeying at join/leave events.
    - Attacker packet rejections.
    - Filtering if the joining node is attacker node

5. Analysis and Visualization

After running NS-3 to generates .csv output files such as keys.csv, rekey_latency.csv, and affectn_log.csv. These files contain key values, rekeying latency, and 1-affects-N metrics.

To process and visualize these results:

1. Install Python and Required Libraries- pip install pandas matplotlib numpy scipy
2. Activate Python Virtual Environment
3. To obtain the Python Graph Script - Save a Python file (e.g., graph_plot.py) with code to read the CSV file and plot the graph:

4. Eg : To check randomness

```
# Load keys from CSV
df = pd.read_csv("keys.csv", header=None, names=["Key"])

# Frequency of each unique key
key_counts = df["Key"].value_counts().sort_index()
total_keys = key_counts.sum()

# Shannon Entropy
entropy = -sum((count / total_keys) * log2(count / total_keys) for count in key_counts)
max_entropy = log2(len(key_counts))
print(f"Shannon Entropy: {entropy:.4f} bits (max = {max_entropy:.4f})")

# Chi-square Test
expected = [total_keys / len(key_counts)] * len(key_counts)
observed = key_counts.tolist()
chi2_stat, p_value = chisquare(f_obs=observed, f_exp=expected)
print(f"Chi-Square Statistic: {chi2_stat:.4f}")
print(f"P-Value: {p_value:.4f}")

# Plot Key Frequency Distribution
plt.figure(figsize=(10, 5))
plt.bar(key_counts.index.astype(str), key_counts.values, color='skyblue')
plt.title("Key Frequency Distribution")
plt.xlabel("Key")
plt.ylabel("Frequency")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Randomness Validation

- Shannon entropy calculation.
- Chi-square test (p-value).
- Graphs showing key value distributions.

Below is the python code :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import chisquare
from collections import Counter
from math import log2

# Load the keys from CSV (no header expected)
df = pd.read_csv("keys.csv", header=None, names=["Key"])

# Count frequency of each unique key
key_counts = df["Key"].value_counts().sort_index()
total_keys = key_counts.sum()



Shannon Entropy
entropy = -sum((count / total_keys) * log2(count / total_keys) for count in key_counts)
max_entropy = log2(len(key_counts))

print(f"\nShannon Entropy: {entropy:.4f} bits (max = {max_entropy:.4f})")

# Chi-square test (comparing to uniform distribution)
expected = [total_keys / len(key_counts)] * len(key_counts)
observed = key_counts.tolist()

chi2_stat, p_value = chisquare(f_obs=observed, f_exp=expected)
print(f"Chi-Square Statistic: {chi2_stat:.4f}")
print(f"P-Value: {p_value:.4f}")

# Plot frequency distribution
plt.figure(figsize=(10, 5))
plt.bar(key_counts.index.astype(str), key_counts.values, color='skyblue')
plt.title("Key Frequency Distribution")
plt.xlabel("Key")
plt.ylabel("Frequency")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Fig 9 :Randomness Test

5.3 Performance Graphs

- Rekeying success rate vs. time.
- 1-affects-N rate comparison
- Rekeying latency plot.

6. Monitoring Security Features

The LGKMP protocol has built-in security monitoring to ensure only legitimate nodes can communicate inside the fog network.

6.1 Attacker Detection and Blocking

- When a node tries to join with a wrong seed or mismatched key, it immediately gets rejected.
- The simulation prints a clear log message in the terminal, for example: [900s] Attacker key mismatch. Packet blocked. This ensures that attackers cannot send or receive valid group messages.

6.2 Seed Leakage prevention:

- **Legitimate Node Key Generation** – A real fog node generates its own its group key using the secure domain seed and node ID .This matches the key expected by the system.
- **Wrong Seed Attack** – If an attacker uses the wrong seed, they get a different key. The system sees the mismatch and blocks the packet.
- **Seed Leak, Wrong Node ID** – Even if the attacker has the correct seed but the wrong node ID, the key will not match, and the attacker is blocked.
- **Full Knowledge Risk** – If an attacker gets both the correct seed and the correct node ID, they can make the correct key. To stop this, the system should hide node IDs and share seeds in an encrypted way.

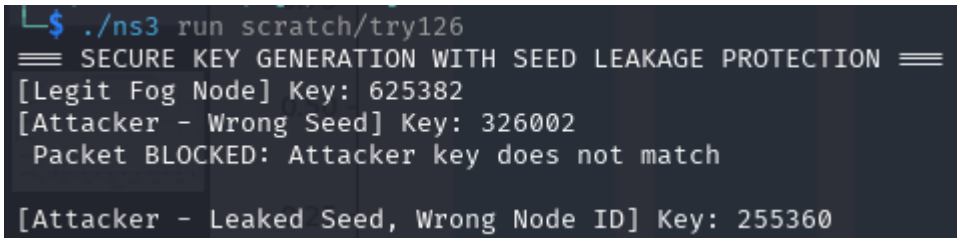


Fig 10 : Seed leakage Test

5.1 Security Log Files

netanim.xml → can be opened in NetAnim to replay and see blocked communications.

- Shows attacker being blocked in NetAnim.
- Shows rekey after a join event.
- Shows packet drop when there is a key mismatch.

```
GNU nano 8.3
scratch/attack.cc *
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/animation-interface.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("ReplayAttackDemo");

void ReplayPacket(Ptr<Node> attacker, Ipv4Address victim, uint16_t port) {

  NS_LOG_UNCOND(" Attacker replaying old packet at " << Simulator::Now().GetSeconds() << "s");

  // Replay same UDP packet as before
  UdpEchoClientHelper fakeClient(victim, port);
  fakeClient.SetAttribute("MaxPackets", UintegerValue(1));
  fakeClient.SetAttribute("Interval", TimeValue(Seconds(1.0)));
  fakeClient.SetAttribute("PacketSize", UintegerValue(64));
  ApplicationContainer replayApp = fakeClient.Install(attacker);
  replayApp.Start(Seconds(5.0));
  replayApp.Stop(Seconds(6.0));
}
```

```
int main(int argc, char *argv[]) {
   Time::SetResolution(Time::NS);

   NodeContainer nodes;
   nodes.Create(4); // 0: Domain, 1: Attacker, 2: Cluster, 3: Fog

PointToPointHelper p2p; p2p.SetDeviceAttribute("DataRate", StringValue("1Mbps"));
   p2p.SetChannelAttribute("Delay", StringValue("5ms"));

   // DomainLeader sends hardcoded "key"
   UdpEchoClientHelper echoClient(i2.GetAddress(1), 9);
   echoClient.SetAttribute("MaxPackets", UintegerValue(1));
   echoClient.SetAttribute("Interval", TimeValue(Seconds(1.0)));
   echoClient.SetAttribute("PacketSize", UintegerValue(64));
   ApplicationContainer clientApp = echoClient.Install(nodes.Get(0));
   clientApp.Start(Seconds(2.0));
   clientApp.Stop(Seconds(3.0));

   // Attacker replays the same packet later
   Simulator::Schedule(Seconds(5.0), &ReplayPacket, nodes.Get(1),
   i2.GetAddress(1), 9);

   Ipv4GlobalRoutingHelper::PopulateRoutingTables();
// DomainLeader sends hardcoded "key"
   UdpEchoClientHelper echoClient(i2.GetAddress(1), 9);
   echoClient.SetAttribute("MaxPackets", UintegerValue(1));
   echoClient.SetAttribute("Interval", TimeValue(Seconds(1.0)));
   echoClient.SetAttribute("PacketSize", UintegerValue(64));
   ApplicationContainer clientApp = echoClient.Install(nodes.Get(0));
   clientApp.Start(Seconds(2.0));
   clientApp.Stop(Seconds(3.0));

   // Attacker replays the same packet later
   Simulator::Schedule(Seconds(5.0), &ReplayPacket, nodes.Get(1),
   i2.GetAddress(1), 9);

   Ipv4GlobalRoutingHelper::PopulateRoutingTables();

   AnimationInterface anim("replay_attack.xml");
   anim.SetConstantPosition(nodes.Get(0), 10, 30);
   anim.SetConstantPosition(nodes.Get(1), 30, 30);
   anim.SetConstantPosition(nodes.Get(2), 60, 30);
   anim.SetConstantPosition(nodes.Get(3), 90, 30);



   anim.UpdateNodeDescription(nodes.Get(0), "DomainLeader");
   anim.UpdateNodeDescription(nodes.Get(1), "Attacker");
   anim.UpdateNodeDescription(nodes.Get(2), "ClusterLeader");
   anim.UpdateNodeDescription(nodes.Get(3), "FogNode");
```

```
anim.UpdateNodeColor(nodes.Get(0), 0, 255, 0);
anim.UpdateNodeColor(nodes.Get(1), 255, 0, 0);
anim.UpdateNodeColor(nodes.Get(2), 0, 0, 255);
anim.UpdateNodeColor(nodes.Get(3), 255, 165, 0);
```

(11)

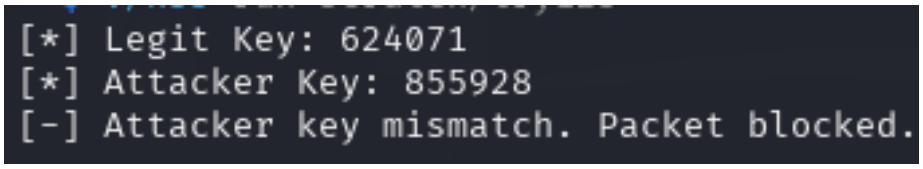


Fig 11 and12: Attacker Node Blocking in LGKMP-Fog